VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

TIL jakožto specifikační jazyk pro inference v PL1
TIL as a specification language for PL1 inferences

2011 Jozef Straka

Prehlásenie študenta:

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave..... .....

Jozef Straka

Poďakovanie:

Touto cestou by som chcel poďakovať pánovi Mgr. Markovi Menšíkovi, Ph.D. za odbornú pomoc pri vedení mojej práce. Zvlášť by som chcel vysloviť poďakovanie pani doc. RNDr. Marie Duží, CSc. za poskytnuté rady a konzultácie.

Abstrakt

Táto diplomová práca sa snaží využiť silu, ktorou oplýva Transparentná intenzionálna logika, pri overovaní správnosti úsudkov. Práca sa zaoberá prevodom konštrukcií z Transparentnej intenzionálnej logiky do predikátovej logiky prvého radu. Cieľom práce bolo navrhnúť a implementovať inferenčný stroj, pre overovanie platnosti úsudkov v predikátovej logike prvého radu s využitím Transparentnej intenzionálnej logiky (TIL) ako špecifikačný jazyk. S tým súvisí i vymedzenie podmnožiny TIL, ktorá je preveditеľná do predikátovej logiky prvého radu.

Kľúčové slová

Transparentná intenzionálna logika, Predikátová logika prvého radu, TIL-Script, Obecná rezolučná metóda.

Abstract

This thesis seeks to use the strength of Transparent intensional logic, to verify the accuracy of judgments. This work deals with the transfer of structures from transparent intensional logic into the first order predicate logic. The goal was to design and implement an inference machine for verifying the validity of judgments in the first order predicate logic with the use of transparent intensional logic (TIL) as specification language. This is related to the definition of a subset of TIL, which is assignable into the first order predicate logic.

Key Words

Transparent intensional logic, First order predicate logic, TIL-Script, General resolution method.

Obsah

1 Úvod

Žijeme v dobe, v ktorej každý upriamuje pozornosť na možné i takmer nemožné výdobytky súčasnej doby a snaží sa o maximálne vyťaženie ponúkaných možností s cieľom vymyslieť čokoľvek, čo by mohlo byť v budúcnosti človeku prínosom. Sme súčasťou spoločnosti preplnenej špičkovými strojmi a prístrojmi, schopnými nahradiť prácu človeka. Jedinou nevýhodou týchto takmer dokonalých strojov je azda to, že zatiaľ nedokážu dýchať a ani zmýšľať ako ľudia. Nechcem byť pesimista, ale myslím že, že to sa im ani nikdy nepodarí. Existujú však nadšenci, ktorý robia všetko preto, aby súčasné stroje pretvorili na inteligentné a prinútili ich aspoň čiastočne zmýšľať ako človek. To docielia prepojením informatiky s vedným oborom – logikou, ktorá je nástrojom zaoberajúcim sa princípmi ľudského uvažovania a zmýšľania. Plynutím času vznikali rôzne pohľady na logiku a tak v dnešnej dobe máme celú radu logík. Vyššie spomínanou problematikou sa zaoberá Transparentná intenzionálna logika (TIL) a predikátová logika. Na rozdiel od predikátovej logiky, ktorá je čisto extenzionálna logika, TIL je jedným z najexpresívnejších logických systémov, ktorý umožňuje rozlišovať medzi intenziami a extenziami (intenzie a extenzie budú definované v kapitole 4.). Práve preto som sa rozhodol využiť TIL pri dokazovaní platnosti úsudkov.

1.1 Cieľ práce

Cieľom tejto diplomovej práce je snaha nájsť a vymedziť takú podmnožinu Transparentnej intenzionálnej logiky, ktorá je prevoditeľná do systému Predikátovej logiky prvého radu. Následne je potrebné navrhnúť a implementovať inferenčný stroj, ktorý bude schopný spracovávať vstupné formule jazyka TIL, prevádzať ich do jazyka predikátovej logiky prvého radu a následne pomocou obecnej rezolučnej metódy overí platnosť zadaných formulí (či záver vyplýva zo zadaných predpokladov, v prípade že vstupom bola iba formula pre záver, program overí či sa jedná o tautológiu).

Výstupom tejto diplomovej práce by mal byť program, ktorý je schopný analyzovať vstupné formule zadané v jazyku TIL, pomocou typovej kontroly overiť formálnu správnosť zadaných formulí, následne overiť, či zadané formule sú prevoditeľné do jazyka predikátovej logiky prvého radu a tieto previesť, a overiť správnosť zadaných formulí s využitím obecnej rezolučnej metódy.

1.2 Štruktúra práce

Aby bolo možné vyhotoviť túto diplomovú prácu, bolo potrebné získať prehľad a základné znalosti z oblastí Transparentnej intenzionálnej logiky a Predikátovej logiky prvého radu. Všeobecný náčrt a popis logických systémov je obsahom druhej kapitoly. Kapitoly 3. a 4. predstavujú obecný základ TIL a PL1. Piata kapitola je výňatkom rozdielov a nedostatkov vlastností, ktorými disponuje PL1 v porovnaní s TIL. Šiesta kapitola nám približuje syntaktický a sémantický pohľad na oznamovacie a opytovacie vety. Ďalšie kapitoly, a to kapitoly 7. a 8. sa zaoberajú vymedzením podmnožiny TIL, ktorú bude možné previesť do systému PL1 a návrhom všeobecných pravidiel, ako postupovať pri prevode.

Vzhľadom k spôsobu zápisu konštrukcií nie je jazyk TIL priamo použiteľný ako počítačovo spracovateľný. Bolo preto potrebné nájsť vhodný interpret, ktorým zapísať vstupné formule. Týmto problémom a popisom upraveného jazyku TIL do počítačovo spracovateľnej podoby – TIL-Scriptu sa zaoberá kapitola 9.

Desiata kapitola sa zaoberá popisom a návrhom riešenia samotnej implementácie programu.

Jedenásta kapitola obsahuje užívateľskú dokumentácia, ktorá slúži ako návod na prácu s vytvorenou aplikáciou.

2 Logické systémy

Logika je vedou o správnosti usudzovania. Z pravidla sa jedná o rozhodnutie správnosti (platnosti) nejakého úsudku (čo je vlastne relace medzi predpokladmi a záverom). Definícia logického vyplývania nám hovorí, že za všetkých podmienok, kedy sú pravdivé premisy (predpoklady), je pravdivý i záver. V bežnom živote človek intuitívne neustále využíva logiky, bez toho, aby si to vôbec uvedomoval. Zavedenie logiky, ako formálneho aparátu pre zaznamenávanie kľúčových elementov v našich výrokoch (kde výrok je tvrdenie, o ktorom má zmysel rozhodovať, či je pravdivý alebo nepravdivý) slúži k výpomoci v situáciách, kedy intuitívne chápanie už tak trochu zaostáva. Teda k tomu, aby mohla byť formálne overená korektnosť nejakého deduktívneho myslenia. To sa overuje na základe dokazovania pravdivosti (alebo platnosti v širšom spektre pojatia) a odvodzovania (alebo vyplývania) z nejakých dopredu stanovených formalizovaných premis (predpokladov).

Pojmom logika sa už odpradávna zaujímajú filozofi, ktorí v podstate položili základy tejto robustnej vede (zakladateľom logiky je podľa súčasných zdrojov grécky filozof Aristoteles). Logika však neostala len predo dvermi filozofov, ale časom si ju obľúbili aj ľudia zameraní na iné vedy – matematici, humanisti a v neposlednej rade taktiež informatici. S časom sa vyvinuli rôzne pohľady na logiku ako vedu a tak v dnešnej dobe máme celú radu logík, začínajúc Aristotelovou, tradičnou, filozofickou, symbolickou formálnou, matematickou a modernou.

V dnešnej dobe pracujeme hlavne s modernou logikou. Jej základy položil nemecký matematik Gottlob Frege na protest tradičnej logiky, ktorá podľa neho vyčerpala svoje možnosti a už dávno mala byť na logickom dôchodku. Na rozdiel od tradičnej logiky, moderná berie samu seba ako vedu o formách a zákonoch správneho usudzovania a teda je schopná riešiť zložitejšie problémy.

Matematické logiky – ako výroková alebo predikátová logika sú logiky formálne, kde pojem „formálne“ znamená fakt, že sa pri formalizácii viet prirodzeného jazyka zameriava výhradne na formu tvrdení (a nijako neskúma jeho obsah). Teda v podstate analyzuje iba štruktúru vety a to striktne analyticky, bez toho, aby akokoľvek bližšie skúmala jej empirický charakter (empirické vlastnosti ich jednotlivých komponent). Teda pracuje iba s jej pravdivostnými hodnotami bez ohľadu na jej význam. Tento úkaz sa nazýva princíp extenzionality a je typickou vlastnosťou klasických logík. Ďalšou vlastnosťou je princíp dvojhodnotovosti. Ten je v podstate viazaný na predchádzajúci princíp a to tým spôsobom, že pravdivostné hodnoty sú vždy len dve – pravda a nepravda.

Logické systémy, ktoré vnímajú ľudské tvrdenia tak, ako bolo uvedené vyššie (teda dodržujú princípy extenzionality a dvojhodnotovosti) sa nazývajú klasické.

Na druhej strane systémy, ktoré tieto dva princípy nerešpektujú (teda sa snažia zachytiť jemnejšiu analýzu tvrdení v prirodzenom jazyku) sa nazývajú neklasické. Výhodou je samozrejme presnejšia analýza. Na druhú stranu, väčšina týchto systémov je stále vo vývoji. Okrem toho, čím viac lingvistických úkazov jazyka sa snažíme zachytiť nejakým formálnym systémom, tým viac sa systém stáva komplexnejším a teda i teoreticky náročnejšie zvládnuteľným.

3 Predikátová logika 1. radu (PL1)

Výrokovou logikou (VL) sa nebudem zaoberať, jej teóriu je možné nájsť v literatúre [1].

Predikátová logika 1. radu je zovšeobecnením klasickej výrokovej logiky a to znamená, že je oproti nej v analýze viet precíznejšia, pretože dokáže oproti výrokovej logike (ktorá v podstate len pomenováva výroky a skladá ich do štruktúr pomocou logických spojok) rozlíšiť subjekt (či už konkrétne indivíduum, zastúpené indivíduovou konštantou, ako napr. človek Peter, tak indivíduovou premennou z nejakého daného oboru) a predikát, čo znamená vlastnosť alebo vzťah, ktorý je subjektu priradený. Vlastnosť priraďuje predikátový symbol nejakému indivíduu. V prípade, že chceme realizovať vzťah, potom predikátový symbol, definujúci vzťah, priraďuje túto reláciu dvojici, či obecne n-tici indivíduí. K indivíduám sa najviac viažu tzv. kvantifikátory – všeobecný a existenčný, ktoré určujú, či existuje aspoň jeden objekt, ktorý má nejakú vlastnosť (v prípade existenčného) alebo či všetky objekty majú nejakú vlastnosť (v prípade všeobecného) v rozsahu daného oboru (množiny). Samozrejme pri analýze súvetí a formalizácii spojok medzi jednotlivými výrazmi (ako konjunkcia, disjunkcia, implikácia a ekvivalencia) potom PL1 ďalej postupuje rovnako ako výroková logika. Oproti nej je teda ale o dosť viac obohatená o zaznamenávanie vlastností a vzťahov medzi indivíduami (realizované predikáty), ktoré tvoria daný obor – nejaké univerzum. Bohužiaľ ani predikátová logika nie je dostatočne jemná na to, aby zaznamenala mnoho špeciálnych úkazov prirodzeného jazyka. Čo však neznamená, že je PL1 menejcenná, pretože jej primárny účel je skôr v oblasti matematiky, kde zohrala významnú úlohu. Aplikácia PL1 na analýzu prirodzeného jazyka je skôr až sekundárnou záležitosťou.

Opäť spomenieme, že PL1 je logika extenzionálna. Pritom realita sa môže meniť. Platnosť úsudkov tu však nezávisí na aktuálnom stave sveta a to je podstatný nedostatok pri analýze prirodzeného jazyka. Okrem toho sa objavujú ďalšie lingvistické úkazy, ktoré PL1 nezachytí (napr. vety hovoriace o neexistujúcich entitách – takéto entity denotujú prázdnu triedu. Porovnajme napr. drak a minotaurus, najväčšie prirodzené číslo a najmenšie záporné celé číslo. Všetky tieto entity denotujú prázdnu triedu). Alebo neúplnosť pojatia viet, resp. extenzionálne poňatie výrazu hovoriace o nejakej entite, ktorou v rôznych okamžikoch zastupuje iné indivíduum – napr. veta „môj sused je povaľač“ (raz to môže byť Peter Saňák, inokedy František Krátky, atď.). V extenzionálnom prístupu, v supozícii *de re* (Rozdiel medzi *de dicto* a *de re* – podrobne vysvetlím v kapitole 5.) hovoríme vždy o konkrétnom indivíduu, i keď očividne predchádzajúce vety o žiadnom konkrétnom indivíduu nehovorili. Ich pravdivosť na Petrovi Saňákovi, Františkovi Krátkom a ďalších bezprostredne nezávisí. S predchádzajúcimi úvahami sa spojuje taktiež spôsob čítania viet. Napr. rozprávame sa o prezidentovi USA. O kom sa rozprávame? Rozprávame sa o konkrétnej osobe za daných okolností, alebo sa rozprávame o danom úrade a nezaujíma nás, kto tento úrad vlastne zastupuje? Poňatie veku ako extenzii najviac prináša značné obmedzenie o hodnotách týchto viet, ktoré z tohto pohľadu môžu nadobúdať len dvoch možností – buď je veta pravdivá, alebo nepravdivá (dvojhodnotovosť). Ako je vidno, predikátová logika skutočne nedokáže zachytiť veľa jazykových úkazov, ktoré sú špecifické pre náš jazyk. Nie je teda príliš vhodným aparátom pre precíznu formalizáciu viet prirodzeného jazyka, avšak prináša tie výhody, že jazyk predikátovej logiky je dotiahnutý do precíznej formy k použitiu v oblasti matematiky.

V rámci výrokovej logiky môže byť formalizovaná len veľmi malá časť úsudkov [1]. Ukážme si to na príklade úsudku, ktorý môžeme na prvý pohľad považovať za správny.

Každý človek robí chyby
Peter je človek

Peter robí chyby

Uvedené tri vety si označíme symbolmi P, Q, R, potom pokus o formalizáciu v rámci výrokovej logiky je daný nasledujúcim úsudkom: P, Q / R, čo odpovedá formule:

$(P \wedge Q) \supset R$

Táto formalizácia je však nedostačujúca. Dôvodom sú:

- Úsudok P, Q / R nie je platný, formula $(P \wedge Q) \supset R$ nie je tautológiou, i keď úsudok demonštrovaný príkladom je evidentne platný.
- Uvedené tri výroky sú z hľadiska výrokovej logiky elementárne a navzájom nezávislé. V skutočnosti však majú vnútorné komponenty, sú štruktúrované a existuje medzi nimi väzba. Termín „človek“ sa vyskytuje vo výrokoch P i Q, termín „robí chyby“ vo výrokoch P i R a termín „Peter“ vo výrokoch Q i R.

Úsudok uvedený na príkladu vyššie môžeme v PL1 formalizovať nasledovne:

$\forall x[P(x) \supset Q(x)], P(p) \models Q(p)$

alebo:

$\{\forall x\,[P(x) \supset Q(x)] \wedge P(p)\} \supset Q(p)$,

kde:

- *x* je predmetová (indivíduová) premenná prebiehajúca určitou predmetnou oblasťou – Universum diskursu,
- *p* je indivíduová konštanta z danej predmetnej oblasti (v uvedenom príklade konkrétny človek Peter),
- *P, Q* sú určité vlastnosti predmetov z univerza diskursu (v uvedenom príklade ich interpretujeme ako vlastnosti mysliacich bytostí „byť človekom“ a „robiť chyby“),
- P(x), Q(x), P(p), Q(p) znamenajú, že x resp. p má vlastnosť P resp. Q,
- zápis $\forall x$ [ ] znamená, že pre všetky indivídua z predmetnej oblasti platí to, čo je uvedené v hranatých zátvorkách.

Predikátová logika 1. radu formalizuje úsudky o vlastnostiach predmetov a vzťahov medzi predmetmi pevne danej predmetovej oblasti (univerza). PL 1. radu je zovšeobecnením výrokovej logiky, ktorú môžeme považovať za logiku nultého radu. Zvyčajne je postačujúca pre formalizáciu mnohých matematických a iných teórií.

V nasledujúcom texte pri popisovaní syntaktických dôkazových metód PL1 som čerpal z literatúry [1], kde nájdete podrobnejšie informácie o spomínanej problematike.

3.1 Obecná rezolučná metóda

Rezolučná metóda je jednou z procedúr (algoritmov), ktoré parciálne rozhodujú, či daná formula PL1 je nesplniteľná. Pre predloženú formulu F, ktorá je nesplniteľná, procedúra zistí túto skutočnosť v konečnom čase a zastaví sa. V prípade, že F je splniteľná, algoritmus nemusí svoju činnosť nikdy skončiť. Ak chceme teda rozhodnúť, či je daná formula F logicky pravdivá, použijeme rezolučnú metódu na negovanú formulu ¬F a zisťujeme, či je nesplniteľná. V prípade, že je tomu tak, procedúra to zistí a vydá kladnú odpoveď. V opačnom prípade proces nemusí nikdy skončiť. Konkrétne, ak chceme zistiť, či $\{A_1, \ldots, A_n\} \vDash B$, aplikujeme rezolučnú metódu na formulu $A_1 \wedge \ldots \wedge A_n \wedge \neg B$, pretože ak je táto formula nesplniteľná, potom je formula $(A_1 \wedge \ldots \wedge A_n) \supset B$ tautológia a vzťah vyplývania platí.

Rezolučnú metódu je možné aplikovať len na formule špeciálneho tvaru, v tzv. klauzulárnej (Skolemovej) forme. Najskôr ukážem, že každú formulu je možno previesť do klauzulárnej formy tak, že výsledná formula je splniteľná, práve keď východisková formula je splniteľná.

3.2 Dokazovanie logickej pravdivosti formule v PL1

Aby sme mohli dokázať logickú pravdivosť formule F v PL1, musíme vykonať nasledovné kroky:

1. formulu F znegujeme
2. formulu ¬F prevedieme do klauzulárnej formy $(\neg F)^S$
3. Na formulu $(\neg F)^S$ postupne uplatňujeme rezolučné pravidlo. Ak získame prázdnu klauzulu, je dôkaz úspešne u konca.

Automatické dokazovanie v predikátovej logike zovšeobecňuje postupy automatického dokazovania výrokovej logiky. Situácia v predikátovej logike je oproti VL zložitejšia a to predovšetkým z týchto dôvodov:

- procedúra prevedenia formule na klauzulárný tvar je komplikovanejšia. Oproti VL obsahuje navyše:
  - prevod formule na prenexní tvar
  - elimináciu kvantifikátorov z formule
- tvar rezolučného odvodzovacieho pravidla je zložitejší. Jeho použitie si vyžaduje tzv. unifikáciu.

3.2.1 Definícia: Prenexní tvar formule

Formula F predikátovej logiky je v prenexním tvaru, ak má podobu

$Q_1 x_1 \; Q_2 x_2 \; \ldots \; Q_n x_n \; B$,

kde:

- $n \geq 0$ a pre každé i = 1, 2, ..., n je Q_i buď všeobecný $\forall$, alebo existenčný $\exists$ kvantifikátor,
- $x_1, x_2, \ldots x_n$ sú navzájom rôzne indivíduové premenné,
- B je formula utvorená z elementárnych formulí, používajúca iba výrokové faktory ¬, $\wedge$, $\vee$.

Výraz $Q_1x_1\ Q_2x_2\ \dots\ Q_nx_n$ sa nazýva *prefix (charakteristika)* a B *otvoreným jadrom (maticou)* formule F v prenexnom tvare.

Každú formulu môžeme prepísať do prenexného tvaru, tzn. ku každej formule predikátovej logiky F existuje formula F* v prenexnom tvare, ktorá je s formulou F ekvivalentná (tzn. F $\Leftrightarrow$ F*).

3.2.2 Definícia: Skolemova forma

Skolemova forma uzavretej formule je prenexní tvar tejto formule, ktorá neobsahuje žiadne existenčné kvantifikátory. Skolemova forma vznikne z prenexnej formy opakovaným použitím nasledujúcich dvoch operácií (skolemizáciou):

$$\forall x_1\ \forall x_2 \dots \forall x_n\ \exists y\ F(x_1, x_2, \dots, x_n\, y) \rightarrow$$
$$\forall x_1\ \forall x_2 \dots \forall x_n\ F\left(x_1, x_2, \dots, x_n, f(x_1, x_2, \dots, x_n)\right)$$

kde f je nový (v jazyku doteraz nepoužitý) n-ární funkčný symbol, tzv. *Skolemova funkcia*,

- $\exists x\ \forall y_1\ \forall y_2 \dots \forall y_n\ F(x, y_1, y_2, \dots, y_n) \rightarrow\ \forall y_1\ \forall y_2 \dots \forall y_n\ F(c, y_1, y_2, \dots, y_n)$,

kde *c* je nová (v jazyku doteraz nepoužitá) indivíduová konštanta, tzv. *Skolemova konštanta*;

- každému eliminovanému existenčnému kvantifikátoru zodpovedá iná Skolemova funkcia alebo konštanta.

Skolemovu formu formule F označíme zápisom F^S.

Literál je atomická formula alebo negácia atomickej formule (napr. $p(f(x))$, $\neg q(y)$).

Klausule je disjunkcia literálov (napr. $[p(f(x)) \vee \neg q(y)]$).

Konjunktívny normálny tvar formule predikátovej logiky je prenexní tvar formule, ktorej matica je konjunkcia disjunkciou literálov (tzn. konjunkcia klauzulí).

Disjunktívny normálny tvar formule predikátovej logiky je prenexní tvar formule, ktorej matica je disjunkcia konjunkcií literálov.

Klauzulárna forma formule je Skolemova forma, ktorej matica je v klauzulárnom tvare, tzn. je konjunkciou klauzulí.

Poznámky:

a) Skolemové konštanty a funkcie predstavujú predmety (reprezentanty predmetov), o ktorých existencii vypovedajú pôvodné formule. Tak napríklad:

- $\exists x \forall y\ A(x, y) \longrightarrow \forall y\ A(c, y)$

  Ak považujeme za univerzum množinu všetkých nezáporných čísiel a realizáciou (interpretáciou) predikátu A je relace < (teda A(x, y) „chápeme ako“ x < y), tak potom c interpretujeme ako 0. V tomto modelu je konštanta c jediná, ale v iných modeloch to tak nemusí byť.

- $\forall x \exists y\ A(x, y) \longrightarrow \forall x\ A(x, f(x))$

  Ak za univerzum považujeme množinu reálnych čísiel a oborom pravdivosti predikátu A je relácia <, tak potom interpretáciou funkčného symbolu f môže byť napríklad f´, ktorá je zadaná predpisom $f(x) = x + \sqrt{3}$.

b) Po vykonanej skolemizácii zostávajú v prefixu formule iba obecné kvantifikátory. Najdôležitejšie pre náš ďalší výklad je klauzularná forma formule:

$\forall x_1 \forall x_2 \ldots \forall x_n [C_1 \wedge C_2 \wedge \ldots C_k]$, kde C_i sú klauzule (disjunkcie literálov). Vzhľadom k tomu, že uvažujeme iba uzavreté formule, nie je nutné tieto kvantifikátory explicitne uvádzať.

c) Skolemova forma F^S uzavretej formule F nie je ekvivalentná s formulou F, ale platí:

$$\models F^S \supset F, \text{ alebo (prípadne) } F^S \models F.$$

Zachováva splniteľnosť.

Veta(Skolem)

Každá formula F môže byť prevedená na takú formulu F^S v klauzulárnej (Skolemovej) forme, ktorá je splniteľná práve keď F^S je splniteľná.

Dôkaz: Uvedieme algoritmus prevodu F → F^S.

1. krok: Utvorenie existenčného uzáveru formule F. (zachováva splniteľnosť)
2. krok: Eliminácia nadbytočných kvantifikátorov (Ekvivalentný krok)
   Z formule F vypustíme všetky kvantifikátory $\forall x_i$, $\exists x_i$, v ktorých rozsahu sa nevyskytuje premenná x_i
3. krok: Premenovanie premenných (Ekvivalentný krok)
   Premenuje všetky premenné, ktoré sú v F kvantifikované viac ako raz tak, aby všetky kvantifikátory mali navzájom rôzne premenné.
4. krok: Eliminácia spojok $\supset, \equiv$ podľa týchto vzťahov (Ekvivalentný krok):
   $(A \supset B) \Leftrightarrow (\neg A \vee B), (A \equiv B) \Leftrightarrow (\neg A \vee B) \wedge (\neg B \vee A)$
5. krok: Presun spojok $\neg$ dovnútra. (Ekvivalentný krok)
6. krok: Presun kvantifikátorov doprava (Ekvivalentný krok)
   Nahradzujeme podľa týchto ekvivalencií (Q je kvantifikátor $\forall$ alebo $\exists$; © je symbol $\wedge$ alebo $\vee$; A, B neobsahujú voľnú premennú x):
   $Qx\,(A \copyright \mathrm{B(x)}) \Leftrightarrow A \copyright \mathrm{Qx\,B(x)}, \mathrm{Qx\,(A(x) \copyright B)} \Leftrightarrow \mathrm{Qx\,A(x) \copyright B}$

   Pozn: Pred vykonaním kroku 7. je vhodné vykonať ekvivalentné zjednodušujúce úpravy formule.
7. krok: Eliminácia existenčných kvantifikátorov (Zachováva splniteľnosť)
   Vykonávame postupne skolemizáciu podformulí $Qx\,B(x), Qx\,A(x)$, ktoré sme získali v predchádzajúcom 6. kroku, teda náhradou existenčne kvantifikovaných formulí formulami bez existenčného kvantifikátoru podľa Definície 3.2.2.
8. krok: Presun všeobecných kvantifikátorov doľava (Ekvivalentný krok, pretože sme už vykonali 3. krok a platí ekvivalencia podľa 6. kroku)
9. krok: Použitie distributivních zákonov. (Ekvivalentný krok)
   Vykonáme postupné náhrady formulí vľavo formulami vpravo:
   $(A \wedge B) \vee C \Leftrightarrow (A \vee C) \wedge (B \vee C), A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

3.2.3 Obecná rezolučná metóda - príklad

Ako som už uviedol vyššie, aby sme mohli na formulu aplikovať rezolučné pravidlo, musíme danú formule najskôr znegovať a previesť do klauzulárnej formy $(\neg F)^S$. Potom môžeme pokračovať uplatňovaním rezolučného pravidla. Avšak v predikátovej logike to nie je také jednoduché, ako vo výrokovej logike. Problémom sú literály a ich termy. Vo výrokovej logike by sme literály s opačným znamienkom mohli vyškrtávať, avšak v predikátovej logike tomu tak nie je. Musíme brať v úvahu rôzne termy.

Ukážeme si to na príklade:
Majme formulu F v klauzulárnej forme:

$$\forall x\,[\neg \mathrm{K}(\mathrm{x}) \vee \neg \mathrm{L}(\mathrm{x})] \wedge \forall y\,[\neg \mathrm{H}(\mathrm{y}) \vee \mathrm{K}(\mathrm{y})] \wedge [H(a) \wedge L(a)]$$

Dokážeme že táto formula je splniteľná. Najskôr si teda vypíšeme jednotlivé klauzule pod seba a pokúsime sa uplatňovať rezolučné pravidlo.

1. $\neg \mathrm{K}(\mathrm{x}) \vee \neg \mathrm{L}(\mathrm{x})$
2. $\neg \mathrm{H}(\mathrm{y}) \vee \mathrm{K}(\mathrm{y})$
3. $H(a)$
4. $L(a)$

Klauzule 1. a 4. obsahujú literály s opačným znamienkom, avšak uplatneniu rezolúcie bráni to, že $\neg \mathrm{L}(\mathrm{x}) \neq L(a)$. Musíme si však uvedomiť, že všetky premenné sú univerzálne kvantifikované, a že platí zákon konkretizácie. Môžeme sa teda pokúsiť nájsť vhodnú substitúciu termu za premennú tak, aby sme dostali zhodné unifikované literály. V príklade teda použijeme nasledujúcu substitúciu:

x/a

Po prevedení substitúcie dostaneme nasledujúce klauzule:

1. $\neg \mathrm{K}(\mathrm{a}) \vee \neg \mathrm{L}(\mathrm{a})$
2. $\neg \mathrm{H}(\mathrm{y}) \vee \mathrm{K}(\mathrm{y})$
3. $H(a)$
4. $L(a)$

Teraz môžeme na klauzule 1. a 4. uplatniť rezolučné pravidlo:

5. $\neg \mathrm{K}(\mathrm{a})$

Následne prevedieme substitúciu y/a na klauzule 2. a 5.

1. $\neg \mathrm{K}(\mathrm{a}) \vee \neg \mathrm{L}(\mathrm{a})$
2. $\neg \mathrm{H}(\mathrm{a}) \vee \mathrm{K}(\mathrm{a})$
3. $H(a)$
4. $L(a)$
5. $\neg \mathrm{K}(\mathrm{a})$

Uplatnením rezolučného pravidla dostaneme:

6. $\neg \mathrm{H}(\mathrm{a})$

A následne uplatnením rezolúcie na klauzule 3. a 6. dostaneme prázdnu klauzulu.

7. □

Uplatnením rezolučného pravidla sme došli ku sporu (prázdnej klauzule), teda zadaný úsudok je platný.

4 Transparentná intenzionálna logika

TIL, alebo transparentná intenzionálna logika je v dnešnej dobe považovaná za najsilnejší aparát pre explikáciu prirodzeného jazyka. Zo subjektívneho pohľadu poznávajúceho vonkajší svet si tento subjekt uvedomuje platné fakty a prostredníctvom nejakého jazyka tieto poznatky zaznamenáva. Vonkajší svet je v tomto prípade množina indivíduí, medzi ktorými sú rozdistribuované vlastnosti a vzťahy (ako u PL1, kde tieto väzby, prípadne vlastnosti realizovali predikátové symboly). Je však potrebné brať v úvahu intenzionálne poňatie týchto vlastností a vzťahov, pretože okrem extenzionálnych faktov (ako sú veci pevne dané – napr. výraz prvočíslo alebo čísla ako také) je potrebné brať v úvahu empirickú flexibilitu. Teda na rozdiel od modelovania vlastností a vzťahov extenzionálnym spôsobom explikácie jazyka (kde každá trieda objektov je pevne daná tým, ktoré objekty do nej patria) intenzionálna explikácia neodmieta empirický náhľad na stav vecí za daných okolností. Z intenzionálneho rámca empirického skúmania plynie celý systém TIL. Aby som to objasnil – ak máme výraz označujúci nejakú matematickú (tzn. analyticky pevne danú) charakteristiku (napr. výraz byť prvočíslom), potom tento výraz denotuje triedu prvočísiel, ktorá je v každom okamžiku konštantná. Na druhú stranu, výraz pre nejakú empirickú vlastnosť, napr. byť auto, denotuje v rôznych okamžikoch rôznu triedu. Táto teória nás navádza k tomu zaviesť pojem „možný svet“, ktorý si predstavíme ako súbor faktov (teda nie súbor vecí – ako definoval Wittgenstein), ktoré môžu platiť. Jeden z týchto „možných svetov“ je aktuálny svet, je to teda súbor všetkých aktuálne platných faktov. Zistiť, ktorý svet je však aktuálny vedie k filozofickým úvahám vševedúcnosti. Preto je TIL postavená na myšlienke, že pre každý časový okamžik existuje množina rôznych možných svetov. Denotát empirických výrazov je teda z hľadiska intenzionálneho prístupu závislý na parametre možných svetov a parametre časových okamžikov. To čo je denotované (stále uvažujeme empirický výraz) je nazývané intenzitou, ktorá je definovaná ako funkcia na parametroch aktuálneho sveta a času. Intenzita je teda funkciou z možných svetov a časov do pravdivostných hodnôt. To čo je denotované sa nazýva intenziou, zatiaľ čo návratová hodnota takejto intenzie (funkcie) v danom svete a aktuálnom čase je nazývaná referenciou (príslušného výrazu). Preto nie je referencia záležitosťou logickej sémantiky, pretože je zistiteľná len na základe skúseností. Na druhú stranu denotát neempirických výrazov je vo všetkých možných svetoch a časových okamžikoch rovnaký. Denotátom neempirického výrazu je teda za všetkých okolností vždy ta istá trieda (teda nemusíme uvažovať nad aktuálnym svetom alebo časom). V uvedenom textu som čerpal poznatky z literatúry [3].

Uvediem príklad:

Chceme zistiť hodnotu intenzie (teda denotát) výrazu „počet planét slnečnej sústavy“ (empirický výraz). Ako bolo uvedené, intenzia je funkciou (v závislosti na aktuálnom svete a čase) a teda vracia nejakú hodnotu špecifickú pre aktuálny svet a aktuálny okamžik. Denotátom výrazu „počet planét slnečnej sústavy“ (tieto riadky píšem v 12:59, 30.3.2011, vonku svieti slnko, dýchame kyslík a prezidentom ČR je Václav Klaus atď.) je funkcia, ktorá vracia číslo 8.

Také jednoduché to zas nebude. Denotát, totiž funkcia, ktorá vracia referenciu intenzie môže byť zadaná nekonečne mnohými spôsobmi, konštrukciami, ktoré sú explikáciami zmyslu.

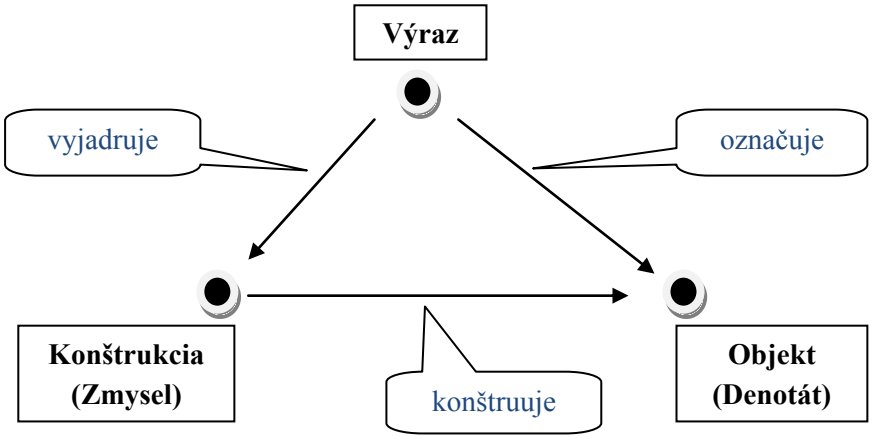
Večernica = Zornica

Výraz Zornica alebo Večernica nedetonuje priamo nejaké indivídua (triedy alebo pravdivostné hodnoty), ale denotuje svoj zmysel (konštrukciu). Výraz Večernice teda nedenotuje Venušu, ale svoj zmysel. Čo keď existuje človek, ktorý vie, že Večernica je (v danom stave vecou) Venušou na oblohe. Je známe, že Večernica je rovná Zornici, avšak takúto skutočnosť už daná osoba nemusí poznať. Avšak keby sme skutočnosť, že Večernica = Zornica zadali ako ďalší predpoklad úsudku (kde prvý predpoklad by bol, že niekto vie, že Večernica je Venušou) a ako záver položili tvrdenie, že niekto vie, že Večernica je Zornica, bol by oprávnene tento úsudok platný? Niekto predsa nemusí vedieť, že Zornica = Večernica, avšak na základe daných predpokladov by takýto úsudok mal byť platný.

Denotačná sémantika teda neposkytuje vždy vhodný prostriedok pre analýzu, náhradou je sémantika zmyslu.

V TIL jednoducho niektoré výrazy denotujú extenziu, iné zasa intenziu. V TIL je teda poňatie Zornice (najjasnejšej hviezdy na oblohe) na rozdiel od Venuše (čo je indivíduum) niečím iným – je niečím, čím Venuša môže byť. Zornica nie je teda nejaké indivíduum, ale akási funkcia, rola, úrad, ktorý môže nejaké dané indivíduum v závislosti na stave vecí (alebo skôr faktov) zastupovať. Zornica teda denotuje funkciu z možných svetov a časov do indivíduí, v aktuálnom svete potom referuje Venušu. Rovnakú referenciu v danom stave sveta a časovom okamžiku poskytuje i výraz Večernica.

Celý tento scenár zachycuje nasledujúca schéma:



Zachytením rozdielu medzi výrazmi definujúcimi extenziu a intenziu však problémy nekončia. Je potrebné prijať ďalšie jemnejšie ohľady pri formalizácii niektorých výrazov, aby sme zamedzili tzv. paradoxu vševedúcnosti.

Dajme tomu, že niekto vie, že 2 + 2 = 4. Vyplýva z toho taktiež že niekto vie, že odmocnina zo 16 = 4 ? Ako môžeme vedieť, že niekto niečo vie, hoci sa to môže zdať ako triviálna skutočnosť. Analytický fakt, že 2 + 2 = 4 je pevne daná skutočnosť, ktorá je pravdivá za všetkých okolností – tautológia. Je teda možné na jej miesto dosadiť akúkoľvek pravdivú vetu. Z toho však nevyplýva, že niekto vie že odmocnina zo 16 = 4. Výrazy 2 + 2 = 4 a odmocnina zo 16 = 4 nemajú rovnaký význam. I k takémuto problému sa TIL stavia čelom a zavádza pojem konštrukcie.

Myslím že nadišiel ten správny čas, aby som ozrejmil a popísal základy Transparentnej intenzionálnej logiky. Začneme teda typmi prvého radu. Spomenuté definície som čerpal z literatúry [2], kde nájdete bližšie a podrobnejšie informácie o spomínanej problematike.

4.1 Typy 1. radu

Od vzniku teórie modelov je zvykom uvažovať domény, ktorých prvky alebo zobrazenia z týchto domén slúžia k interpretácii jazykových výrazov. Prostredie TIL pre kategorizáciu týchto entit používa rozvetvenú hierarchiu typov. Jej základnými jednotkami sú objekty atomických typov. Skutočnosť, že objekt O je typu α budeme zapisovať ako O je α-objekt, alebo O/α. Objektová báza predstavuje špeciálny typ bázy, nad ktorým môže byť vybudovaný systém funkcií a ich konštrukcií. V tomto systéme modelujeme konceptuálne schéma. Objektová báza TIL je tvorená štyrmi základnými množinami:

- množina pravdivostných hodnôt, ktorú predpokladá už výroková logika (Typ ο). Najznámejší typ, ktorý nadobúda hodnôt **pravda, nepravda**.
- predikátová logika obecne predpokladá množinu indivíduí (taktiež nazývaná aj ako univerzum diskurzu) (Typ ι)
- množina časových okamžikov – reálnych čísiel (Typ τ)
- množina možných svetov – prvky logického priestoru (Typ ω)

4.1.1 Definícia: Epistémická báza

Báza v TIL je tzv. epistémická báza, čo predstavuje množinu {ο, ι, τ, ω}, kde ο je množina pravdivostných hodnôt, ι je univerzum diskurzu a jeho prvky sú indivídua, τ je množina časových okamžikov (reálnych čísiel) a ω je množina možných svetov.

4.1.2 Definícia: Typy 1. radu

Nech B je báza, tzn. množina vzájomne disjunktných neprázdnych množín. Tak potom:

- Každý prvok báze B je atomický (elementárny) typ 1. radu nad B.
- Nech α, β_1, ..., β_n ($n > 0$) sú typy 1. radu nad B. Potom množina ($\alpha\beta_1...\beta_n$) všetkých n-árnych parciálnych funkcií, tzn. zobrazení z $\beta_1 \times ... \times \beta_n$ do α, je molekulárny (alebo tiež funkcionálny) typ 1. radu nad B.
- Nič iného nie je typom 1. radu nad B, okrem týchto predchádzajúcich bodov.

4.1.3 Intenzie, extenzie

Empirické výrazy (vrátane viet) sú také výrazy, ktoré označujú (netriviálne) *intenzie,* čo sú funkcie z možných svetov a časových okamžikov do istého typu α. Hodnotu označenej intenzie, tzn. *referent* výrazu v určitom svete a čase, nie je možné určiť logickými prostriedkami. Napríklad empirická veta: „Počet planét (našej slnečnej sústavy) je 8“ má v aktuálnom svete a čase pravdivostnú hodnotu pravda. Toto však neplatí obecne v iných časoch, prípadne iných svetoch. Intenzie sú parciálnymi funkciami, preto môžu byť na niektorých argumentoch nedefinované – nevracajú hodnotu).

Neempirické výrazy (hlavne matematické a logické výrazy) oproti tomu typicky označujú *extenzie.* Sú to také objekty, ktoré nie sú funkcie z možných svetov alebo časov (ich modálna alebo temporálna podmienenosť nemusí byť braná v úvahu).

Takže:

(α-)intenzie sú prvky typu $((\alpha\tau)\omega)$, (budeme skracovať ako $\alpha_{\tau\omega}$), teda funkcie z možných svetov do ľubovoľného typu α.

(α-)extenzie sú objekty typu α, kde $\alpha \neq (\beta\omega)$ pre žiadny typ β. Teda extenzie sú α-objekty, ktorých doménou nie je množina možných svetov.

Typickými príkladmi intenzií sú:

- ***Vzťahy*** medzi prvkami typov $\beta_1,...,\beta_n$ sú zobrazenia typu $(o\beta_1...\beta_n)_{\tau\omega}$
  - Napríklad vzťah „byť starší ako“
- ***Propozície*** sú zobrazenia typu $o_{\tau\omega}$.
  - Označujú sa vetami. Napr. „Bratislava je hlavné mesto Slovenska.“
- ***Vlastnosti indivíduí*** sú objekty typu $(o\iota)_{\tau\omega}$.
  - Napríklad: „byť mužom“, „byť chudobný“, „byť študentom“
- ***Individuové úrady*** sú objekty typu $\iota_{\tau\omega}$.
  - Príkladom sú objekty označené výrazmi: „prezident Slovenskej republiky“, „najväčšia hora“. Líšia sa od vlastností (ako napr.: „byť prezidentom SR“) tým, že v závislosti na svetoch a časoch nevracajú množinu indivíduí, ale najviac jedno indivíduum.
- ***Vlastnosti propozícií*** sú objekty typu $(oo_{\tau\omega})_{\tau\omega}$.
  - Napríklad: „byť pravdivá“, „byť nutná“
- ***Empirické funkcie – atributy*** sú objekty typu $(\alpha\beta)_{\tau\omega}$.
  - Príkladom môže byť „vek daného indivídua“, „deti danej osoby“, „otec niekoho“, „prezident niečoho“.

4.2 Konštrukcie

Konštrukcie sú abstraktné procedúry, ktoré na základe daných vstupov po konečnom počte krokov dodávajú určitý výstup. Je to akýsi „návod“ ako po konečnom počte krokov získať požadovaný výstup (entitu). Jednotlivé kroky tohto návodu môžu byť opäť iba procedúry (inštrukcie). Tento návod však v sebe musí zahŕňať aj ostatné (neprocedurálne) objekty, aby celá procedúra mohla na základe vstupov identifikovať výstupný objekt.
Následne spomeniem štyri základné typy konštrukcií.

Prvým základným typom sú *atomické* konštrukcie, ktoré identifikujú vstupné objekty priamo, bez pomoci iných inštrukcií. Sú to *premenné* a *trivializácia*.

Ďalšie dva typy konštrukcií sú už viac štrukturované. Tieto obsahujú ako časti aj iné inštrukcie, ktorých výstupy používajú na konštrukciu výstupného objektu. Sú to *kompozícia*, ktorá špecifikuje spôsob aplikácie funkcie na argument, a *uzáver*, ktorý udáva ako pomocou lamda-abstrakcie vytvoriť funkciu.

4.2.1 Premenné

Najjednoduchším typom konštrukcií sú premenné. Premenné a konštrukcie, ktoré obsahujú premenné konštruujú objekty určitého typu v závislosti od valuácie. Hovoríme že premenná *v*-konštruuje, kde *v* je parameter valuácie. Pre každý typ máme k dispozícii spočítateľne nekonečne veľa premenných. Premenné zvykneme označovať ako (x, y, z, x_1, y_1, ...).

Valuácia je funkcia, ktorá každej premennej priradí presne jedno indivíduum, a to – pre premennú x_i – práve to indivíduum, ktoré je na i-tej pozícii ($v(xi)=\xi$).
Napríklad pre univerzum U={A, B} máme sekvenciu AAAA..., BAAAA..., pričom v_1 priraďuje x_1 A, x_2 taktiež A, x_3 taktiež A, ..., v_2 priraďuje x_1 B, x_2 A, x_3 A, atď.

4.2.2 Trivializácia

Najjednoduchšou zloženou konštrukciou je trivializácia.
Definícia: Nech X je nejaký objekt alebo konštrukcia, tak potom ^{0}X je konštrukcia nazývaná trivializácia. Trivializácia ^{0}X konštruuje X bez akejkoľvek zmeny.

Každý objekt, dokonca aj konštrukcia, môže byť trivializovaná. Trivializácia slúži ako okamžitá konštrukcia, predstavuje jednokrokovú procedúru. Táto konštrukcia nám umožňuje rozpoznať medzi entitami a spôsobom ako sú konštruované. Ak X je nejaká konštrukcia, ^{0}X konštruuje toto X, nie objekt vytvorený touto konštrukciou.

Trivializáciu môžeme použiť v prirodzenom jazyku. Napríklad objekt „slnko“ konštruujeme ako ^{0}slnko. Rovnako môžeme konštruovať objekty ľubovoľného typu. Napríklad číslo 2 predstavuje τ-objekt. Konštrukcia ^{0}2 konštruuje tento objekt (číslo 2) bez akejkoľvek zmeny.

4.2.3 Kompozícia

Kompozícia predstavuje operáciu aplikácie funkcie *f* na n-ticu argumentov $A_1,...,A_n$ za účelom získania hodnoty funkcie *f* na týchto argumentoch. Ako príklad si môžeme zobrať funkciu sčítania čísiel 1 a 3. Čísla 1 a 3 predstavujú objekty typu τ. Operátor funkcie sčítania + je objekt typu $\tau\tau\tau$. Výsledkom aplikácie funkcie sčítania na čísla 1 a 3 (1+3) je číslo 4. Všetko objekty musia byť trivializované. Výsledná konštrukcia vyzerá nasledovne: [^{0}+ ^{0}1 ^{0}3].

Konštrukcia ^{0}4 dáva rovnaký výsledok, ako predchádzajúca konštrukcia, a to číslo 4. Z toho vyplýva, že pre TIL nie je dôležitý len výsledok, ale aj spôsob, akým sme ho dostali. Jeden objekt sme schopný zapísať pomocou viacerých konštrukcií, ktoré sa však od seba odlišujú.

Definícia: Kompozícia

Nech X je konštrukcia, ktorá v-konštruuje funkciu $F/(\alpha\ \beta_1\ ...\ \beta_n)$ a nech $X_1, ...,X_n$ sú konštrukcie, ktoré v-konštruujú entity $b_1/\beta_1,...,b_n/\beta_n$. Potom $[X\ X_1...X_n]$ je konštrukcia nazývaná kompozícia (alebo tradične aplikácia). Ak funkcia F nie je definovaná na n-ticu objektov $b_1, ...\ b_n$, potom kompozícia $[X\ X_1 \ldots X_n]$ je v-nevlastná (tzn. nekonštruuje nič). Inak v-konštruuje hodnotu F na $<b_1,...,b_n>$.

4.2.4 Uzáver

Ďalší zo spôsobov vytvárania funkcií je inšpirovaný λ-kalkulom. Uzáver je konštrukcia duálna ku kompozícii. Umožňuje nám vytvárať funkciu, a tak vhodným spôsobom hovoriť o celej funkcii (o jej priebehu), nielen o jej hodnote pre daný argument.

Definícia: Uzáver

Nech $x_1, ..., x_n$ sú navzájom rôzne premenné a X konštrukcia. Potom $[\lambda x_1\ ...\ x_n X]$ je konštrukcia nazývaná uzáver (alebo tradične abstrakcia), ktorá v-konštruuje nasledujúcu funkciu F: Nech v' je valuácia, ktorá priraďuje premenným x_i objekty b_i ($1 \leq i \leq n$) a inak je rovnaká ako valuácia v. Potom ak X je v'-nevlastná, je F nedefinovaná na $<b_1,...,b_n>$. Inak je hodnotou funkcie F na $<b_1,...,b_n>$ objekt v'-konštruovaný konštrukciou X.

Napríklad matematické vyjadrenie funkcie f: $(x_1 + x_3)$ môžeme prepísať nasledovne:
$\lambda\ x_1, x_3\ [^0+\ x_1\ x_3]$.
Alebo $\lambda\ x_1,\ x_2\ [^0<\ x_1\ x_2]$ konštruuje funkciu porovnania dvoch čísiel, ktorej výsledkom je pravdivostná hodnota (pravda, nepravda). Je to funkcia, ktorá priraďuje hodnotu pravda, ak je prvé číslo menšie ako druhé, v opačnom prípade priradí hodnotu nepravda.

4.2.5 Prevedenie a dvojité prevedenie

Idea konštrukcií je založená na λ-kalkulu - ako redukcia všetkých operácií k vytvoreniu funkcií a použiteľnej funkcii k argumentom. Pridaním nevyhnutých zastávok, ako napr.: premenných a trivializácií, dostaneme štyri druhy konštrukcií ako sme si definovali vyššie. Tu sa vynára prirodzená otázka. Môžeme byť spokojný s týmito štyrmi druhmi? V skutočnosti definujeme ďalšie dva druhy konštrukcií, a to prevedenie a dvojité prevedenie (exekúcia, dvojitá exekúcia).

Pre akúkoľvek entitu X môžeme tiež hovoriť o konštrukcii prevedenie (e*xekúcia)* X a označovať ju ako ^{1}X. Ak X je konštrukcia, potom ^{1}X je X. Naopak, ak X nie je konštrukcia, potom ^{1}X je (nevydarená) konštrukcia, ktorej začiatočným bodom je X, a ktorá nevracia nič (konštrukcia nemôže byť vykonaná). Teda, ak X je v-nevlastná konštrukcia, alebo nie je konštrukcia, ^{1}X je v-nevlastná, inak v-konštruuje to, čo je konštruované pomocou X.

Ak konštrukcia X konštruuje ďalšiu konštrukciu, môžeme po vyhodnotení konštrukcie X pokračovať a vyhodnotiť taktiež konštrukciu konštruovanou X. Túto dvojkrokovú konštrukciu budeme nazývať *dvojitou exekúciou* X, budeme ju označovať ako ^{2}X. Pre akúkoľvek konštrukciu X, konštrukcia ^{2}X je v-nevlastná, ak samotná X nie je konštrukciou, alebo ak v-nekonštruuje

konštrukciu, alebo ak v-konštruuje v-nevlastnú konštrukciu. V opačných prípadoch ^2X v-konštruuje to, čo je v-konštruované tým, čo je v-konštruované pomocou X.

Uveďme si jednoduchý príklad.
Nech X je číselná premenná, $x \rightarrow_v \tau$. Exekúcia ^1x v-konštruuje rovnaké číslo ako x, dvojitá exekúcia ^2x je v-nevlastná pre akékoľvek v (x v-nekonštruuje žiadnu konštrukciu). Teda konštrukcia $^2(^1x)$ je v-nevlastná, zatiaľ čo $^2(^0x)$ v-konštruuje to isté číslo ako x.

Len ťažko môžeme nájsť miesto, v ktorom by boli tieto dva druhy konštrukcií relevantne využiteľné. Sme presvedčený, že nie sú potrebné.

Exekúcia slúži predovšetkým za účelom rozlišovania medzi konštrukciami a nekonštrukciami, ktoré však možno dosiahnúť nezávisle na sebe.

Dvojitá exekúcia – čokoľvek môže byť vykonané týmto druhom konštrukcie, môže byť vykonané pomocou príslušnej funkcie *F* tak, že zadaním konštrukcie *C* a valuácie *V*, funkcia *F* použitím *C*, vráti objekt (ak existuje), ktorý je v-konštruovaný pomocou C.
Preto teda nepoužívame tieto dodatočné druhy konštrukcií.

4.3 Typy vyššieho radu

V prirodzenom jazyku sa nevyjadrujeme len o objektoch 1. radu. Jednotlivé pojmy – konštrukcie nemusia byť vždy len používané, ale aj spomínané. K tomuto nám však nebude postačovať jednoduchá teória typov, ktorá nie je dostatočne silná. Potrebujeme zachádzať s konštrukciami ako s plnohodnotnými objektmi a práve preto musíme definovať konštrukcie a typy vyšších rádov.

Vyššie rády sú definované niekoľkými krokmi. Následne si definujme konštrukcie radu *n* a potom, pomocou nich, typy radu *n* +1.

Definícia: Konštrukcia radu *n*

Nech α je typ radu *n*, tak potom:

- nech *x* je premenná, ktorá v-konštruuje objekty typu radu *n*. Potom *x* je konštrukcia radu *n*,
- nech *X* je *α*-objekt. Potom ^0X je konštrukcia radu *n*,
- nech *C* je kompozícia $[X\,X_1 \ldots X_m]$ a *n* je taký najvyšší rád, že aspoň jedna z konštrukcií $X, X_1, \ldots, X_m$ je konštrukciou radu *n*, potom *C* je konštrukcia radu *n*,
- nech *C* je uzáver $[\lambda x_1 \ldots \lambda x_m\, X]$ a *n* je najvyšší rád taký, že aspoň jedna z konštrukcií $x_1, \ldots, x_m, X$ je konštrukcia radu *n*, potom C je konštrukcia radu *n*,
- nič iného nie je konštrukcie radu *n*.

Definícia: Typy radu *n* + 1

Nech $*_n$ je množina všetkých konštrukcií radu *n*.

- $*_n$ je typ radu *n* + 1,
- nech *n* + 1 je taký najvyšší rád, že aspoň jeden z typov $\alpha, \beta_1, ..., \beta_m$ je typ radu *n* + 1, tak potom $(\alpha, \beta_1, ..., \beta_m)$ je typ radu *n* + 1,
- nič iného nie je typ radu n + 1.

Ukážme si to na jednoduchom príklade.

Nech x je číselná premenná v-konštruujúca objekt typu τ. Keďže τ je typ 1. radu, bude aj x konštrukcia 1. radu. Takže x je prvok ^{*1} a teda patrí do typu 2. radu. Zoberme si konštrukciu αx. Pretože x je konštrukcia (objekt) typu 2. radu, bude aj αx konštrukcia 2. radu. Takže je prvok ^{*2} a teda patrí do typu 3. radu.
Môžeme písať:

$x \rightarrow_v \tau,\ x/^*1$
$\alpha x \rightarrow_v {}^*1,\ \alpha x/^*2$

4.4 Logická analýza

Analýza výrazu spočíva v objavovaní logickej konštrukcie zakódovanej pomocou výrazu. K dispozícii poskytujeme vždy výrazy s ich procedurálnymi štrukturálnymi významami, ktoré sú vložené ako TIL konštrukcie.
TIL metóda analýzy sa skladá z troch krokov:

1. *Typovo-teoretická analýza*, tzn. priradenie typov objektom, ktoré sú zmienené (spomenuté) v analyzovanej vete.
2. *Syntéza*, tzn. kombinácia konštrukcie objektov s cieľom zostrojiť návrh typu $o_{\tau\omega}$, označujúci celú vetu.
3. Typovo-teoretická kontrola – nepovinný krok, ktorý pomocou spätnej kontroly typov určí, či je výsledný typ objektu, ktorý konštrukcia vracia, správny.

Ukážeme si to na príklade
Budeme analyzovať vetu: „Primátor Ostravy je múdry“.

Veta hovorí o úrade primátora mesta Ostravy a o vlastnosti byť múdry, ktorá je pripisovaná jednotlivcovi (ak existuje), ktorý zastáva tento úrad. Predpokladom je, že ostravský primátor existuje, tzn. že úrad je obsadený. Ak nie je, potom tvrdenie označíme ako vetu, ktorá nemá žiadnu pravdivostnú hodnotu. Táto skutočnosť musí byť odhalená našou analýzou. Ukážem ako:

1. Typovo-teoretická analýza:

   Primátor_čoho / $(\iota\iota)_{\tau\omega}$
   Ostrava / ι - je indivíduum,
   Primátor_Ostravy / $\iota_{\tau\omega}$ - predstavuje indivíduový úrad,
   Múdry / $(o\iota)_{\tau\omega}$ - je vlastnosť indivídua.

2. Syntéza:

Teraz musíme spojiť konštrukcie objektov, ktorých typovú analýzu sme vykonali v 1. kroku, s cieľom zostaviť tvrdenie typu $o_{\tau\omega}$, ktoré označuje celú vetu (propozíciu). Najjednoduchšou konštrukciou z vyššie uvedených objektov sú ich *trivializácie*:

$$^0Primátor\_čoho,\ ^0Ostrava,\ ^0\text{Múdry}.$$

Atribút *Primátor_čoho* musí byť najskôr extenzionalizovaný cez kompozíciu $^0Primátor\_čoho_{wt}$ a potom aplikovaný na Ostravu:

$$[^0Primátor\_čoho_{wt}\ ^0Ostrava]$$

Nakoniec zostavením (výrazu) nad svetom w a časom t, získame úrad:

$$\lambda w \lambda t\ [^0Primátor_{wt}\ ^0Ostrava]$$

Ale vlastnosť byť múdry nemôžeme pripisovať k jednotlivým kanceláriám. Skôr to môžeme pripísať indivíduu obsadzujúcemu úrad. Takže úrad musí byť podrobený intenzionálnemu zostupu do v-konštrukcie indivídua obsadzujúceho úrad (ak existuje):

$$\lambda w \lambda t\ [^0Primátor\_čoho_{wt}\ ^0Ostrava]_{wt}.$$

Samotný objekt musí byť taktiež extenzionalizovaný:

$$^0Múdry_{wt}.$$

Kompozíciou týchto dvoch konštrukcií získame pravdivostnú hodnotu True, False, alebo nič, podľa toho, či primátor mesta Ostravy je alebo nie je múdry, alebo neexistuje. Napokon zostavením (výrazu) nad *w, t* konštruujeme tvrdenie:

$$\lambda w \lambda t\ [^0Múdry_{wt}\ \lambda w \lambda t\ [^0Primátor\_čoho_{wt}\ ^0Ostrava]_{wt}].$$

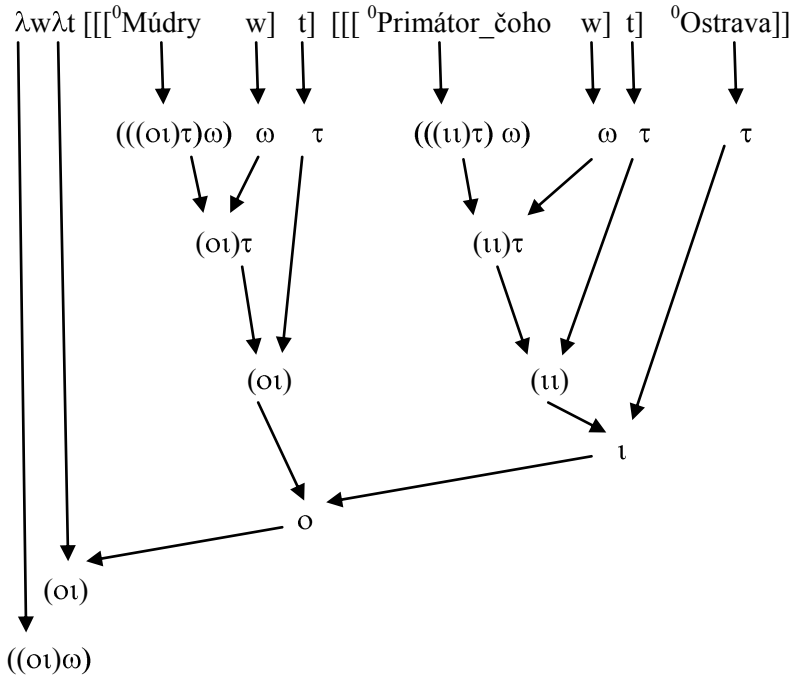
Tento Uzáver môžeme ekvivalentne zjednodušiť na:

$$\lambda w \lambda t\ [^0Múdry_{wt}\ [^0Primátor\_čoho_{wt}\ ^0Ostrava]]$$

V každom svete a v každom čase ($\lambda w \lambda t$) postupujeme nasledovne: Za prvé zistíme, kto je primátor mesta Ostravy . Ak tam nie je, potom skončíme s prázdnou pravdivostnou hodnotou. V opačnom prípade zistíme, či získaný jedinec má vlastnosť byť múdry. Ak áno, tak je výsledkom True, inak False.

3. Typovo-teoretická kontrola:

   V tomto kroku kreslíme typovo-teoretický štrukturálny strom, pomocou ktorého môžeme skontrolovať, či určité prvky z vyššie uvedeného uzáveru sú spojené správnym spôsobom.



Výsledkom typovej kontroly je typ ($o_{\iota\omega}$). Výsledný typ teda súhlasí. Konštrukcia je po typovej stránke správna.

5. Rozdiely medzi PL1 a TIL

V tejto časti by som rád zhrnul vlastnosti, ktoré sú v podstate obmedzením klasického logického systému PL1 oproti „neklasického“ intenzionálneho systému TIL.
V nasledujúcich bodoch je zhrnuté, ktoré z logických prvkov poskytovaných TIL sa nevzťahujú na tradičné systémy (ako je aj PL1):

1. Extenzionálne systémy predikátovej logiky (akéhokoľvek radu – PL(n)) nedokážu rozlišovať medzi analytickými a empirickými výrazmi, ktoré menia svoje vnímanie (svoju konštrukciu) podľa stavu vecí. Denotát (označený objekt) z pohľadu TIL sa mení v závislosti na okolnostiach a čase. PL1 nedokáže zohľadniť modálne alebo temporálne aspekty. To znamená, že v rámci predikátovej logiky nedokážeme empirické výrazy riadne analyzovať. Extenzionálna logika pracuje iba s intenziami 0. radu (extenziami). Zavedenie intenzií nenulových rádov (intenzií) umožňuje pracovať s možnými svetmi. Napríklad tvrdenie: „teplota je 20 stupňov Celzia a stúpa“ sa nedá v extenzionálnej logike analyzovať ako (teplota=20)∧stúpa(teplota), pretože z toho je možné dokázať, že stúpa(20), čo nie je významom tvrdenia.
   Problémy ako modalita (ktorá zavádza nové výrokové modifikátory ako výrazy obsiahnuté vo vetách typu „je možné že ...“, „nutne je pravda, že ...“, „je pravdepodobné, že ...“, ...) a temporalita (ktorá zohľadňuje časové okamžiky), z ktorých ani jeden nie je schopný systém PL zachytiť, môžu intenzionálne logiky (ako napríklad TIL) efektívne ošetriť.
2. Predikátová logika prvého radu neumožňuje analyzovať rozdiely medzi užitím a zmieňovaním funkcií (vlastností a vzťahov). Taktiež nedokáže správne zachytiť rozdiel medzi použitím výrazu (a jeho príslušného významu, tzn. konštrukcie) v supozícii *de dicto* a *de re*. Tento problém by potenciálne mohli vyriešiť predikátové logiky vyšších rádov.

   Príslušný výraz F vo vete V je v supozícii de dicto, ak pravdivostná hodnota danej vety V vo svete w a čase t nezávisí na hodnote funkcie (intenzie) označenej výrazom F v tomto svete w a čase t. Príslušná funkcia sa akoby nevykonáva (príkladom môže byť výraz: „Môj sused je často chorý“. Môjho suseda nepoznám, ale nech tam býva ktokoľvek, je to môj sused a je často chorý. Príslušná funkcia je teda užitá v supozícii de dicto).

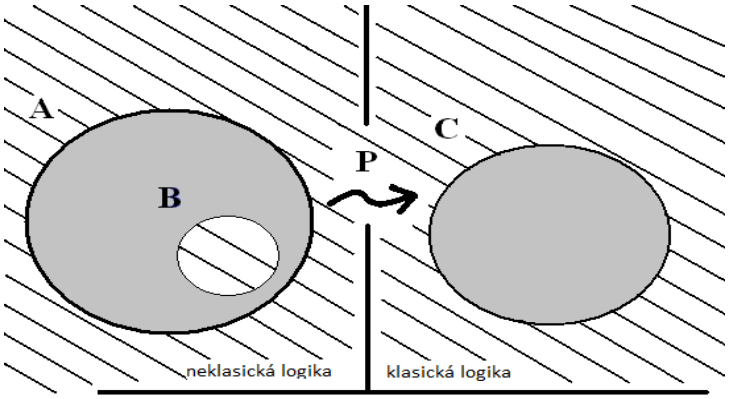
   Príslušný výraz F vo vete V je v supozícii de re, ak referencia výrazu E (teda hodnota intenzie – označenej funkcie) má vplyv na pravdivostnú hodnotu vety v danom svete w a čase t (Príkladom môže byť výraz: „Môj sused je často chorý“ – poznám ho, je náchylný na choroby a preto býva často chorý).
3. Denotačný prístup k sémantike (významom výrazu je koncipovaný ako denotovaný / označený objekt bez ohľadu na spôsob, akým je tento objekt prezentovaný; v prípade empirických výrazov je denotátom intenzia – funkcia z možných svetov) neumožňuje rozlíšiť synonymné výrazy od ekvivalentných. Z tohto dôvodu je riadna analýza tzv. hyperintenzionálnych kontextov (znalostí, domnienok, vedení a hypotéz) kameňom úrazu pre všetky denotačné sémantiky, kedy príslušné logické konštrukcie nielen užívame, ale taktiež zmieňujeme. Tento problém by mohlo vyriešiť použitie procedurálnej (štrukturálnej) sémantiky.
4. Formalistický prístup k sémantike neumožňuje efektívne riešiť detailnejšie rozdiely medzi reprezentáciou konštrukcie (formula v PL1 v podstate vyjadruje schéma – „označuje“

množinu – konštrukcií) a vlastnou konštrukciou. Problém by mohlo vyriešiť použitie transparentného prístupu.

5. Klasické systémy predikátovej logiky nie sú schopné spracovávať parciálne funkcie – tzn. výroky bez pravdivostnej hodnoty. Tento rys ich hlavne obmedzuje v prípade, kedy analyzujeme prázdny pojem (napríklad „najväčšie prvočíslo“), teda rozumieme danému výrazu, avšak neexistuje pre neho konkrétny zástupca v realite, preto nie sme schopný rozhodovať o pravdivosti alebo nepravdivosti tvrdení. Lepším príkladom je problém existencie francúzskeho kráľa: „Francúzsky kráľ je holohlavý“. V súčasnom svete a čase nemôže byť toto tvrdenie ani pravdivé, ani nepravdivé. Keby sme totiž dokázali priradiť jednoznačne pravdivostnú hodnotu tomuto tvrdeniu, vyplývala by z neho existencia francúzskeho kráľa. Problémy (ne)existencie a vety s existenčnými presupozíciami sú ďalším kritickým bodom u klasických logických systémov.

Úvaha:

Uvažujme množinu A ako základnú množinu všetkých vlastností logického systému TIL a množinu B, reprezentujúcu vlastnú podmnožinu týchto vlastností základnej množiny A, ktoré boli uvedené v bodoch 1 – 5. Ďalej uvažujeme množinu C reprezentujúcu súhrn všetkých vlastností logického systému PL1. Potom sme schopný vytvoriť taký transformačný algoritmus (prevod) P, ktorý prevedie určité vlastnosti z množiny B (tie, ktoré sú preveditеľné) do množiny C. Vymedzením podmnožiny TIL, ktorá je preveditеľná do systému PL1, sa budem zaoberať neskôr, v ďalších kapitolách.



Obrázok č. 1: Náčrt problému transformačného algoritmu a vymedzenia podmnožiny

Z predchádzajúceho výkladu je zrejmé, že pri prekladu formulí z TIL do PL1 je potrebná redukcia výnimočnosti špecifikovanej v rozvinutom systému TIL. Hlavným problémom pri prevodu je potom spätné zavedenie princípu dvojhodnotovosti a extenzionality špecifickej práve pre PL1.

5.1 De dicto / De re

V nasledujúcej časti by sme si mali objasniť základné rozdiely medzi užitím výrazu v supozícii *de dicto* a *de re*. Tieto dva výrazy používame k označovaniu dôležitých rozdielov v intenzionálnych tvrdeniach (ide hlavne o rozdiely v domnienkových postojoch). Ako bolo spomenuté vyššie, systém PL1 neumožňuje analyzovať rozdiely medzi užitím a zmieňovaním funkcií (vlastností a vzťahov), čiže nedokáže správne zachytiť rozdiel medzi použitím výrazu (a jeho príslušného významu, tzn. konštrukcie) v supozícii *de dicto* a *de re*. Tento problém by potenciálne mohli vyriešiť predikátové logiky vyšších rádov, akou je aj TIL.

Definujme si teda tieto dva pojmy:

De dicto

Príslušný výraz *F* je v supozícii *de dicto* vo vete *V*, ak pravdivostná hodnota danej vety *V* vo svete *w* a čase *t* nezávisí na hodnote funkcie (intenzie) označenej výrazom *F* v tomto svete *w* a čase *t*. Inými slovami, príslušná funkcia je iba zmieňovaná a nie je použitá pre získanie svojej hodnoty (Pojem, ktorý je významom výrazu *F* je použitý – príslušná konštrukcia túto funkciu konštruuje).

De re

Príslušný výraz *F* je v supozícii *de re* vo vete *V*, ak referencia výrazu *E* (hodnota označenej funkcie) ovplyvňuje pravdivostnú hodnotu vety. Presnejšie povedané, pravdivostná hodnota vety v danom svete *w* a čase *t* závisí na hodnote funkcie (intenzie) v tomto svete *w* a čase *t*.

Mali by sme ešte poznamenať, že veta je sama v sebe vždy *de dicto*, i keď to z uvedenej charakteristiky nevyplýva.
Klasickým jednoduchým príkladom je dvojica viet:

Americký prezident je demokrat. (*de re*)

$$\lambda w \lambda t[{}^{0}\text{Demokrat}_{wt}\ {}^{0}\text{Americký\_prezident}_{wt}]$$

Americký prezident je voliteľný. (*de dicto*)

$$\lambda w \lambda t[{}^{0}\text{Voliteľný}_{wt}\ {}^{0}\text{Americký\_prezident}]$$

6. Oznamovacie a opytovacie vety

Podľa zámeru hovorcu voči poslucháčovi sa jazyk člení na vety oznamovacie, rozkazovacie a opytovacie. Z logického hľadiska je sémantický obsah oznamovacej (alebo rozkazovacej) vety rovnaký, ako obsah vety opytovacej.
Uvediem príklad:

Jozef je študent.
Je Jozef študent?

Syntaktický rozdiel nám je určite zrejmý, avšak rozdiel v sémantike týchto dvoch viet zaniká. Prvá veta nám oznamuje, že Jozef je študent. Je to nejaký výrok, teda tvrdenie, o ktorom má zmysel rozhodnúť, či je pravdivé alebo nepravdivé. Druhá veta „Je Jozef študent?“ sa nás (na rozdiel od prvej vety) na niečo pýta, avšak rovnako ako u predchádzajúceho tvrdenia, tu stále existuje zmysel rozhodnúť, či je to pravda alebo nie. Pri analýze obidvoch viet je podstatný iba vetný podmet a prísudok – teda objekt (indivíduum) o ktorom veta vypovedá, a jeho vlastnosť (predikát). V praxi by potom tieto vety mali v jazyku TIL rovnakú analýzu, rovnako tak aj v PL1.

Analýza vety v TIL:

$$\lambda w \lambda t[{}^0\check{S}tudent_{wt}\ {}^0Jozef]$$

Analýza vety v PL1:

$$\check{S}tudent(Jozef)$$

Predchádzajúci príklad zohľadňoval otázky typu áno – nie (sme na ne schopný jednoznačne odpovedať).

Obecne môžeme definovať, že opytovacie správy so sémantickým obsahom, konštruujúce tvrdenia typu o_{tw} (propozície), reprezentujú áno – nie otázky, ktorých pravdivostná hodnota je typu (o) a závisí na danom svete *w* a čase *t*.

Okrem týchto základných áno – nie otázok, existujú otázky, ktoré môžeme doplniť o existenciu osôb s nejakými vlastnosťami. V takomto prípade nebude odpoveďou PL1 iba jednoduché áno - nie, ale odpoveď je doplnená o indivídua, ktoré majú tu danú vlastnosť. Takže áno – nie otázky, týkajúce sa existencie jednotlivcov, splňujúcich dané podmienky, sú preložené do PL1 „Wh-otázok“ zodpovedajúcich konštrukcii príslušného objektu.
Ukážme si to na príklade. Majme opytovaciu vetu:

Sú niektorí študenti múdry?

Najskôr si analyzujme vetu pomocou TIL. Typovo teoretická analýza by vyzerala nasledovne:

Študent / $(o\iota)_{\tau\omega}$
Múdry / $(o\iota)_{\tau\omega}$
x / ι

Výsledná konštrukcia vety v TIL je:

$$\lambda w \lambda t [{}^0\exists \lambda x\ [[{}^0\check{S}tudent_{wt}\ x] \wedge [{}^0M\acute{u}dry_{wt}\ x]]]$$

Analýza vety v PL1:

$$\exists x\ [\check{S}tudent\ (x)] \wedge [M\acute{u}dry\ (x)].$$

PL1 sa snaží uspieť pri odvodzovaní týchto dvoch cieľov. Za týmto účelom hľadá znalostnú bázu pre fakta a pravidlá obsahujúce vlastnosti „študent“ a „múdry“, a ak je to možné, zjednocuje premennú *x* s príslušnými argumentami nájdených viet (podmienok) . V našom prípade, možnou odpoveďou môže byť „*Áno, X = Peter*“, za predpokladu, že Peter je múdry a je aj študentom, alebo „*Nie*“, za predpokladu, že neexistuje nikto, kto by bol študent a zároveň múdry.

Pri zostavovaní „wh-otázok“ sa pýtame sa na α-entity vlastniace α-vlastnosť, alebo uspokojujúce α-vzťah v intenzii v danom svete *w* a čase *t*. Analýza je riadená podľa možných odpovedí. Ak je odpoveďou množina jednotlivcov, konštruujeme vlastnosť jednotlivcov. Obecne platí, ak je odpoveďou množina n-tic $\alpha 1,\ldots,\alpha_m$-objektov, konštruujeme vzťah v intenzii typu $(o\alpha 1 \ldots \alpha_m)\tau\omega$.

Uvediem ešte jeden príklad. Majme opäť dve vety, jednu oznamovaciu a druhú opytovaciu:

Ľudia vyšší ako 180cm.
Ktorí ľudia sú vyšší ako 180 cm?

Ani tu však nie je rozdiel v tom, o čom vety vypovedajú, aký je ich sémantický výklad. Je pragmatický, teda vecný obsah stále zachováva a to tým, že obidve vety referujú k nejakým indivíduám z tej istej množiny, pričom vlastnosť, ktorú danej entite priraďujú je taktiež stále rovnaká.

Keďže je TIL špecifikačný a opytovací jazyk zároveň, problémy s prevodom formulí (či už s významom opytovacím, alebo oznamovacím) do PL1 by nemali obmedzovať schopnosť zachytiť sémantiku pôvodných viet.
Skúsme teda analyzovať vyššie spomenutú vetu „Ktorí ľudia sú vyšší ako 180 cm?“.

V TIL by vyzerala konštrukcia takto:

$$\lambda w \lambda t\ \lambda x\ [[{}^0\check{C}lovek_{wt}\, x] \wedge [{}^0> [{}^0V\acute{y}\check{s}ka_{wt}\, x]\ {}^0180]],$$

kde Výška je atribút (empirická funkcia) typu $(\tau\iota)_{\tau\omega}$. K tomu aby sme preložili túto vedu do PL1, musíme atribút Výška konvertovať a binárnu relaci. Ukážem ako:

$$\lambda w \lambda t\ \lambda x\ [\lambda y\ [[{}^0\check{C}lovek_{wt}\, x] \wedge [[y = [{}^0V\acute{y}\check{s}ka_{wt}\, x]] \wedge [{}^0> y\ {}^0180]]]]$$

Tento uzáver transformujeme do PL1 nasledovne:

$$\check{C}lovek(x) \wedge V\acute{y}\check{s}ka(y,x) \wedge (y > 180)$$

Funkčný program by vrátil hodnotu vyššie uvedenej vlastnosti zaznamenanej v danom stavu znalostí, tzn. množinu jednotlivcov, PL1 odpoveď v prípade úspechu v plnení týchto cieľov bude napríklad: „Áno, x= Ježík, y = 185“.

7. Špecifikácia podmnožiny TIL prevediteľnej do PL1

Aby bolo možné vymedziť podmnožinu konštrukcií TIL, ktorá je prevediteľná do PL1, musíme sa najskôr zamyslieť nad vetami (výrazmi), s ktorými budeme pracovať. Systém predikátovej logiky je založený čisto na tom, že každá formula je pravdivá alebo nepravdivá. Na základe pravdivosti premis potom overujeme platnosť či neplatnosť úsudku. Avšak u TIL nás nezaujímajú len pravdivostné hodnoty alebo vyplývanie. TIL je zložitý systém, ktorého cieľom je čo najefektívnejšie analyzovať prirodzený jazyk, a ten sa vo všetkých prípadoch nedá ohodnotiť nulou a jednotkou (pravdou a nepravdou). Vstupom teda môžu byť iba také konštrukcie viet alebo tvrdení, u ktorých sme schopný určiť ich pravdivostnú hodnotu. Čiže sú to vety, na ktoré môže odpovedať buďto True, v prípade že sa jedná o pravdivé tvrdenie, alebo False, v prípade že je dané tvrdenie nepravdivé. Takéto vety rozdeľujeme do dvoch skupín, a to na vety intenzionálne a na vety extenzionálne. Vety majúce intenzionálny kontext sú také, ktorých pravdivostná hodnota je vždy závislá od aktuálneho sveta (ω) a času (τ). Ich pravdivostná hodna je typu $o_{\tau\omega}$. Jedná sa o propozície. Druhým typom sú výrazy majúce extenzionálny kontext. To sú také konštrukcie, ktorých pravdivostná hodnota nie je závislá od aktuálneho stavu sveta alebo času. Ich pravdivostná hodnota je typu o. Najčastejšie vystupujú ako konštrukcie matematických výrazov, vzťahov a funkcií.

7.1 Kvantifikátory

Kvantifikátory vystupujú ako symboly, slúžiace pre vyjadrenie miery prítomnosti danej vlastnosti (predikátu) v istej triede objektov. Rozlišujeme dva základne typy kvantifikátorov, a to všeobecný ($\forall$) a existenčný ($\exists$). Kvantifikátory určujú, či je predikát aplikovateľný aspoň na nejaké jedno indivíduum (v prípade existenčného), alebo na všetky individua (v prípade všeobecného) v rozsahu danej triedy objektov (daného oboru).

Pri prevodu kvantifikátorov z TIL do predikátovej logiky nedochádza k žiadnym problémom. Kvantifikátory vystupujú rovnako v TIL ako aj v PL1.

7.2 Výrokové logické spojky

Rovnako ako v predikátovej logike, aj v TIL sa vyskytujú logické spojky. Účel použitia je rovnaký. Tak ako sa v predikátovej logike spájame dva výroky pomocou logickej spojky a výsledkom je opäť výrok, tak aj v TIL slúžia logické spojky pre spájanie rôznych konštrukcií.

V TIL vystupujú tieto logické spojky: konjunkcia ($\wedge$), disjunkcia ($\vee$), implikácia ($\supset$), ekvivalencia ($\equiv$) a negácia ($\neg$). Pri prevodu nedochádza k žiadnym nezhodám, a tak v PL1 vystupujú tieto spojky rovnako ako v TIL.

7.3 Matematické výrazy

Matematické výrazy zaraďujeme do množiny konštrukcií extenzionálnych, majú extenzionálny kontext. Ich výsledkom je pravdivostná hodnota, ktorá nie je závislá na svete (ω). Celá táto množina matematických vzťahov a funkcií je prevediteľná do predikátovej logiky prvého radu.

Ako príklad uvediem vzťah dvoch čísiel: 10 je viac ako 5.

Zápis v TIL:

$[^0\text{Viac}\ {}^0 10\ {}^0 5]$

Zápis v PL1:

Viac(10,5)

7.4 Vety

Aby bolo možné previesť vety do predikátovej logiky, musíme najskôr určiť, o aký typ vety sa jedná. Propozície môžem rozdeliť na skupinu s výskytom hyperintenzionálneho kontextu a skupinu bez tohto výskytu. Následne si podrobne rozoberieme oba typy viet a ich možný prevod do predikátovej logiky prvého radu.

7.5 Intenzionálne vety

Intenzionálne vety sú také, ktorých pravdivostná hodnota je vždy závislá od aktuálneho sveta (ω) a času(τ). Intenzie predstavujú prvky typu ((ατ)ω). Najdôležitejšími typmi intenzí sú:

- Propozície (zobrazenia typu $o_{\tau\omega}$)
  - Napr.: Jozef je študent
- Vlastnosti indivíduí (objekty typu $(o\iota)_{\tau\omega}$)
  - Napr.: byť bohatý, byť šťastný
- Vzťahy (objekty typu $(o\iota\ \iota)_{\tau\omega}$)
  - Napr.: byť vyšší ako niekto, mať rád niekoho
- Úrady (objekty typu $\iota_{\tau\omega}$)
  - Napr.: prezident ČR, najvyššia hora
- Role (atributy typu $\iota_{\tau\omega}$)
  - Napr.: prezident niečoho, otec niekoho
- Vlastnosti úradov (objekty typu $(o\iota_{\tau\omega})_{\tau\omega}$).

7.6 Vlastnosti indivíduí

Vlastnosti indivíduí sú vlastnosti funkcie, v danom prípade sú to funkcie, ktoré možným svetom a časom priradia triedy objektov, ktoré túto vlastnosť majú (v príslušnom svete *w* a čase *t*). Byť študentom, byť červený sú teda funkcie. Sú to objekty typu $(o\iota)_{\tau\omega}$. Tieto vlastnosti indivíduí sú prevoditeľné do predikátovej logiky prvého radu.

Majme vlastnosť *V* typu $(o\iota)_{\tau\omega}$ a indivíduum *x*, ktoré má túto vlastnosť. TIL konštrukcia vyzerá nasledovne: $[^0V_{wt}\ x]$. Potom do predikátovej logiky prvého radu prevádzame predikátom odpovedajúcu vlastnosť *V: V(x)*.

Uvediem príklad. Majme vetu: Jozef je študent.

Zápis v TIL vyzerá nasledovne:

$$\lambda w\lambda t[^0\check{S}tudent_{wt}\ ^0Jozef]$$

Prevod do PL1 spočíva v tom, že vlastnosť byť študentom prevedieme na predikátový symbol a indivíduum Jozef, ktoré má túto vlastnosť prevedieme na term, v tomto prípade na premennú.

Výsledný zápis v PL1:

Študent(Jozef)

Situácia však môže byť úplné iná, keď k vlastnosti indivídua pribudne nejaká rola. Ako príklad uvediem vetu: Petrov otec je profesor.
Zápis v TIL vyzerá nasledovne:

$$\lambda w\lambda t[^0Profesor_{wt}\ [^0Otec_{wt}\ ^0Peter]]$$

Tu už je prevod do PL1 zložitejší. Na prvej pozícii je vlastnosť byť profesorom. Tú prevedieme na predikát. Otázkou ostáva čo s kompozíciou $[^0Otec_{wt}\ ^0Peter]$. Otec zastáva nejakú funkciu, funkciu byť otcom Petra. Takže otca prevedieme na funkciu. Ostáva nám už iba Peter, ktorého prevedieme na konštantu.

Výsledný zápis v PL1:

Profesor(otec(peter()))

7.7 Vzťahy

Vzťahy nám konštruujú objekty typu $(o\iota)_{\tau\omega}$), kde vstupom sú dve indivíduá, ktoré sú spolu v nejakom vzťahu, a výstupom je pravdivostná hodnota (true, false). Tak isto ako vlastnosti, aj vzťahy indivíduí sú prevoditeľné do predikátovej logiky prvého radu.

Ako príklad uvediem vetu: Karol má rád Máriu.
Zápis v TIL:

$$\lambda w \lambda t[{}^0\text{MáRád}_{wt}\ {}^0\text{Karol}\ {}^0\text{Mária}]$$

Výsledný zápis v PL1:

MáRád(Karol, Mária)

7.8 Úrady a vlastnosti úradov

Majme vetu Karol sa chce stať prezidentom ČR. Súčasným prezidentom ČR je Václav Klaus. Niektorí z nás by si mohli túto vetu vyložiť tak, že Karol sa chce stať Václavom Klausom. To však nie je pravda. Táto veta nám hovorí len o tom, že Karol chce zastávať úrad prezidenta ČR.
Skúsme si túto vetu analyzovať:
Zápis v TIL:

$$\lambda w \lambda t[{}^0\text{ChceSaStať}_{wt}\ {}^0\text{Karol}\ {}^0\text{Prezidentom}]$$

Túžba ChceSaStať po prekladu do PL1 zastáva pozíciu predikátu, a Karol a Prezident budú konštanty. Výsledný zápis po prevedení do PL1 vyzerá nasledovne:

ChceSaStať(karol(), prezidentom())

Ukážme si ešte jeden príklad vety želacej. Majme vetu: Karol chce byť bohatý.
Typová analýza tejto vety vyzerá nasledovne:

Chce/ $(o\iota(o\iota)_{tw})_{tw}$
Karol/ ι
Bohatý/ $(o\iota)_{tw}$

Zápis v TIL:

$$\lambda w \lambda t[{}^0\text{Chce}_{wt}\ {}^0\text{Karol}\ {}^0\text{Bohatý}]$$

Túto vetu je možné preložiť do PL1 a to nasledovne: Z vety jasne vyplýva, že Karol chce mať vlastnosť „byť Bohatý“. To že chceMaťVlastnosť preložíme do PL1 ako predikát, a Karla a vlastnosť byťBohatý preložíme ako konštanty. Týmto prevodom sa však stráca jemnosť analýzy, ktorou oplýva TIL. Pri prevode sa indivíduum i vlastnosť stávajú konštantami.

Výsledný zápis v PL1:

Chce(karol(), bohatý())

Všeobecne je platné, že úrady a vlastnosti úradov sú prevediteľné do PL1, avšak pri prevodu veľakrát dochádza k strate časti informácie, ktorou disponuje veta v TIL. Ak je nejaká vlastnosť prevádzaná v postoji, prevádza sa táto vlastnosť ako konštanta. V inej vete, kde nevystupuje v postoji, to môže byť predikát.
Názorne to môžete vidieť na vyššie spomenutej vete: „Karol chce byť bohatý“. V tomto prípade vystupuje vlastnosť byť bohatý ako konštanta.

No môžeme mať úplne iný príklad. Majme vetu: „Karol je bohatý“.
Tu vystupuje Karol ako indivíduum, ktoré disponuje vlastnosťou, a to byť bohatý. V tomto prípade vlastnosť prevedieme na predikát, a Karla na premennú.
Zápis v TIL:

$$\lambda w \lambda t[{}^0\text{Bohatý}_{wt}\ {}^0\text{Karol}]$$

Zápis v PL1:

Bohatý(Karol)

Lepšie si môžeme ukázať túto stratu časti informácii na príklade: „Karol chce, aby bol Tom bohatý“. V TIL môžeme odvodiť, že Karol chce, aby bol niekto bohatý, ale v PL1 to už nepôjde.
Zápis v TIL:

$$\lambda w \lambda t[{}^0\text{Chce}_{wt}\ {}^0\text{Karol}\ \ \lambda w \lambda t\ [{}^0\text{Bohatý}_{wt}\ {}^0\text{Tom}]\ ]$$

Túto konštrukciu prevedieme do PL1 v tvare:

Chce(Karol(), Aby_Tom_bol_bohaty()).

Chce preložíme ako predikát, Karol po prevode predstavuje konštantu, a zvyšná časť vety: „aby bol Tom bohatý“ sa celé prevedie do druhej konštanty.

7.9 Hyperintenzionálne vety

Typickým príkladom hyperintenzionálneho kontextu sú vety, vyjadrujúce tzv. propozičné postoje. V TIL tento hyperintenzionalizmus označujeme pomocou trivializácie.
Príkladom môže byť veta: Jozef počíta 2+3. Mnohý z nás by mohli odvodiť, že Jozef počíta 5. O tom ale táto veta nevypovedá. Veta hovorí že počíta 2 + 3. Keby sme namiesto toho dosadli číslo 5, zmenili by sme konštrukciu celej vety. A práve preto, aby sme zabránili nechceným nedorozumeniam, musíme použiť trivializáciu, ktorá nám konštrukciu 2 + 3 uzavrie. Celá táto konštrukcia uzavretá trivializáciou bude vystupovať ako jedna entita.

V TIL môžeme túto vetu zapísať nasledovne:

$$\lambda w \lambda t[{}^0\text{Počíta}_{wt}\ {}^0\text{Jozef}\ {}^0[{}^0+\ {}^0 2\ {}^0 3]]$$

Pri prevodu do PL1 musíme dať pozor. Tieto hyperintenzie budeme prevádzať ako konštantu. To znamená, že celá konštrukcia, ktorá je uzavretá trivializácou predstavuje konštantu.

Prevod do PL1 vyzerá nasledovne:

Počíta(Jozef, f()).

7.10 Substitúcie

TIL v sebe zahŕňa možnosť uvoľniť premenné viazané trivializáciou, použitím špeciálnej funkcie SUB, ktorá realizuje substitúciu skutočných aktuálnych hodnôt za formálne parametre. Substitúciu môžeme definovať ako funkciu, ktorá ma tri vstupné parametre:

- Konštrukciu, ktorú budeme substituovať,
- Premennú, za ktorú budeme substituovať,
- Konštrukciu, v ktorej sa bude substituovať,

a výsledkom je nová konštrukcia. Môžeme povedať, že substitúcia je objekt typu $(*_n*_n*_n*_n)$, kde n je rada konštrukcií, ktoré budeme substituovať.

Uvediem príklad: Nech $z \rightarrow \tau$, tak potom konštrukcia $[^0SUB\ ^0[^0+\ ^02\ ^010]\ ^0z\ ^0[^0-\ ^0z\ ^01]]$ je konštrukcia 2. radu a konštruuje konštrukciu prvého radu: $[^0-\ [^0+\ ^02\ ^010]\ ^01]$.

Substitúcie ***nie sú*** previediteľné do predikátovej logiky prvého radu.

7.11 Premenné

Premenné, ktoré sú premennými v TIL, sú rovnako previediteľné na premenné do predikátovej logiky prvého radu.

7.12 Lambda

Majme príklad. Existuje nejaké indivíduum x, ktoré má vlastnosť byť študentom.

Zápis v TIL:

$$\lambda w \lambda t\ [^0\exists \lambda x\ [^0Študent_{wt}\ x]]$$

Pri prevode do PL1 sa eliminujú výskyty lambdy. Použiteľnosť na možný svet a možný čas ($\lambda w \lambda t$) sa pri prevode úplne vynechá, a pri definovaní premenných, v tomto prípade premennej x, sa lambda vynechá, a premenná ostane zachovaná.

Výsledný zápis v PL1:

$$\exists x(Študent(x))$$

Obecne platí, že ak definovaniu premennej pomocou lambdy predchádza jeden z kvantifikátorov (existenčný, alebo všeobecný), pri prevode sa vynechá len lambda, premenná ostane zachovaná. V opačnom prípade sa vynecháva aj premenná.

8. Mapovanie pri prevodu TIL do PL1

Teraz v skratke navrhneme všeobecné schéma pre TIL spracovávanie a prevod jednotlivých jednoduchých konštrukcií do predikátovej logiky prvého radu, podľa typov subjektov, ktoré prijímajú uvedené zložky konštrukcie.

Pracujeme s *de re* nie s *de dicto*. Jazyk PL1 upravíme tak, že výraz, za ktorým budú nasledovať prázdne guľaté zátvorky (), bude špecifikovať konštantu.

Prvým krokom, ktorý musíme vykonať, je eliminácia parametrov ω a τ.
Pre účely analýzy prirodzeného jazyka sa využíva objektová báza:

- ο - množina pravdivostných hodnôt
- ι - množina indivíduí (taktiež nazývaná Universum diskursu)
- τ - množina časových okamžikov (alebo reálnych čísel)
- ω - množina logických možných svetov

Pri prevode je nutné obmedziť bázu na prvé dva typy uvedené vyššie – teda iba na množinu pravdivostných hodnôt a množinu indivíduí.

Aby sme zachovali princíp extenzionality u PL1, je potrebné intenzie v TIL vyradiť. To do istej miery urobí samotný TIL, ktorý jednotlivé intenzie extenzializuje na daný svet a čas, v ktorom má byť vyhodnotená pravdivosť. Tento fakt zachovávajú práve parametre pre daný svet a časový okamžik - τ a ω.
V ďalšej časti používam upravené algoritmy, ktoré boli pôvodne navrhnuté na prevod TIL do Prológu [4].

8.1 Zložky propozíc

Nech P je množina zložiek propozíc uzáveru konštrukcie TIL a $P \rightarrow o$,
potom $\lambda w \lambda t\, P \Rightarrow P$.

Ako príklad uvediem propozíciu:

Jozef je študent.

Prevod TIL do PL1:

$$\lambda w \lambda t\, [{}^0\text{Student}_{wt}\ {}^0\text{Jozef}] \ \Rightarrow\ \text{Student(Jozef)}$$

8.2 Zložky α-vlastností

Nech P je množina zložiek propozíc a $P \rightarrow (o\alpha)_{\tau\omega}$; a α je výsledkom funkcie z možných svetov a časových okamžikov, a $x \rightarrow \alpha$, potom $\lambda w \lambda t\ [P_{wt}\ x] \Rightarrow P(x)$.

Modifikátory sú prevedené na vlastnosti.

Ako príklad uvediem vetu:

Monika je múdry študent.

Prevod TIL do PL1:

$$\lambda w \lambda t\ [^0\text{MúdryŠtudent}_{wt}\ ^0\text{Monika}] \ \Rightarrow\ \text{MúdryŠtudent(Monika)}$$

8.3 Zložky (αβ)-atribútov

Nech $Attr \rightarrow (\alpha\beta)_{\tau\omega}$, a P je množinou zložiek propozíc $P \rightarrow (o\alpha)_{\tau\omega}$; b/β; $\alpha \neq o$; $x \rightarrow \beta$; $y \rightarrow \alpha$.

Zložka je používaná v správe „Inform“ (oznamovacia veta), ktorej obsah je vo forme $\lambda w \lambda t\ [P_{wt}\ [Attr_{wt}\ b]]$.

Tak potom:

$$\lambda w \lambda t\ [P_{wt}\ [Attr_{wt}\ b]] \Rightarrow \lambda w \lambda t\ [^0\forall \lambda y\ [[^0= y\ [Attr_{wt}\ b]] \supset [P_{wt}\ y]]] \ \Rightarrow p(Y) \supset attr(Y,b).$$

Uvediem príklad:

Majme vetu Karlov otec je profesor.

Prevod TIL do PL1:

$$\lambda w \lambda t\ [^0Profesor_{wt}\ [^0Otec_{wt}\ ^0Karol]] \Rightarrow$$
$$\lambda w \lambda t\ [^0\forall \lambda y\ [[^0= y\ [^0Otec_{wt}\ ^0Karol]] \supset [^0Profesor_{wt}\ y]]]$$
$$\Rightarrow \text{profesor(Y)} \supset \text{otec(Y,karol)}.$$

Alebo je tu ešte jeden možný prevod:

$$w \lambda t\ [^0Profesor_{wt}\ [^0Otec_{wt}\ ^0Karol]] \Rightarrow\ \text{Profesor(otec(karol()))}.$$

8.4 Zložky vzťahov v intenziách

Nech $Rel \rightarrow (o\alpha\beta)_{\tau\omega}$; $a \rightarrow \alpha$; $b \rightarrow \beta$.

Tak potom $\lambda w \lambda t\ [Rel_{wt}\ a\ b] \Rightarrow \text{Rel(a, b)}$.

Ukážme si to na príklade:

Karol ma rád Máriu.

Prevod TIL do PL1:

$$\lambda w \lambda t\ [^0MaRád_{wt}\ ^0Karol\ ^0Mária] \Rightarrow \text{MaRád(karol, mária)}.$$

8.5 Zložky matematických vzťahov

Nech $Rel \rightarrow (o\tau\tau)$; a, b/τ; $x \rightarrow \tau$; $y \rightarrow \tau$.

Ak je zložka používaná v správe "inform" (oznamovacia veta) tak potom

$[Rel\ a\ b] \Rightarrow$ Rel(a,b).

Ukážme si to na príklade:

Šesť je viac ako päť.

Prevod TIL do PL1:

$[^0Viac\ ^06\ ^05] \Rightarrow$ Viac(6, 5).

8.6 Zložky matematických funkcií

Nech $Calc \rightarrow (\tau\tau)$; x, $y \rightarrow \tau$; m/τ.
Tak potom: $[Calc\ ^0m] \Rightarrow$ calc(m).

Všeobecne môžeme povedať, že zložky n-árnych matematických funkcií sú prevediteľné do PL1 nasledovne:

Nech $Calc^n/(\tau\tau\ldots\tau)$.
Tak potom $[Calc\ x_1 \ldots x_n] \Rightarrow$ calc(x1,…,xn).

Ukážme si jednoduchý príklad:

Odmocnina zo 144

Prevod TIL do PL1:

$[^0OdmocnicaZo\ ^0144] \Rightarrow$ odmocninaZo(144).

9. TIL-SCRIPT

Vo vytváranom inferenčnom stroji bolo potrebné nájsť vhodný jazyk, akým zapísať vstupné konštrukcie formulí a predpokladov. Mnohí by si mohli myslieť, že je to zbytočné, veď formule budú zapísané v jazyku TIL. Ale vzhľadom k spôsobu zápisu konštrukcií nie je jazyk TIL priamo použiteľný ako počítačovo spracovateľný jazyk pre popis obsahu správ. Konkrétnymi dôvodmi sú:

- notácia zloženia jazyka nebola v TIL plne normalizovaná.
- nebolo možné zakódovať zloženie jazyka iba za pomoci ASCII znakov. Abeceda jazyka TIL zahŕňa aj ďalšie znaky, ako sú napríklad horné indexy, písmena gréckej abecedy apod.
- TIL nešpecifikuje rozhranie ontológie. Obsah jazyka musí byť obmedzený na používanie pojmov danej domény. Tieto pojmy sú definované v ontológii.
- Typový základ jazyka nebol najvhodnejší pre multi-agentové systémy. Hlavným nedostatkom boli niektoré veľmi časté programové typy, ako sú celé čísla, zoznamy a reťazce.

Práve tieto dôvody boli podnetom pre vznik jazyka TIL-Script, ktorý predstavuje variantu TIL upravenú tak, aby ju bolo možno počítačovo spracovať. TIL-Script teda odstraňuje vyššie spomenuté nedostatky jazyka TIL.

TIL-Script je mocný programovací jazyk založený na transparentnej intenzionálnej logike (TIL). Z formálneho hľadiska je TIL čiastočne hyper-intenzionální lambda kalkul.

Jazyk TIL je založený na bázových typoch (o, ι, τ, ω), pretože sú vhodné pre analýzu prirodzeného jazyka. TIL-Script mierne upravil a rozšíril túto množinu bázových typov. Hlavné nedostatky spočívali v spojení časových okamžikov a reálnych čísel v jednom typu a v absencii typu pre celé čísla. Kompletná báza TIL-Scriptu je zobrazená v nasledujúcej tabuľke.

Tabuľka č. 1: Bázové typy v TIL-Scriptu

TIL	TIL-Script	Popis
o	o, Bool	Množina pravdivostných hodnôt (True, False)
ι	i, Indiv	Množina jednotlivcov (indivíduí)
τ	t, Time	Chronológia času
ω	w, World	Množina možných svetov
–	Int	Množina celých čísel
τ	Real	Množina reálnych čísel
–	String	Typ reťazec
α	Any	Nedefinovaný, ľubovoľný typ

9.2 Konštrukcie

V TIL-Scriptu je možné používať všetky druhy konštrukcií, ktoré pozná TIL, výnimkou ostáva *prevedenie (exekúcia)*.

Premenné

Premenná predstavuje reťazec znakov, ktorý začína malým písmenom a musí obsahovať iba alfanumerické znaky (písmena, číslice) a znak podtržítko („ _ “).
Príklad premenných: *študent, stôl, drevený_stôl*

Trivializácia

Trivializácia sa označuje symbolom 0 v hornom indexu („ ^{0} “). V TIL-Scriptu sa namiesto tohto znaku používa symbol apostrof. („ ‘ “). Akýkoľvek objekt môže byť predmetom trivializácie. Napríklad trivializácia objektu *P* by sa v TIL-Scriptu zapísala ako *‘P*

Uzáver

V TIL-Scriptu je syntaxe pre uzatvorenie podobná pôvodnému TIL. Ten nahradzuje symbol λ symbolom spätné lomítko „ \ “. Taktiež sa používa symbol „ ‘ “ pre trivializáciu. Okrem toho všetky typy musia byť definované. Uzáver v TIL-Scriptu potom vyzerá nasledovne:

[\x:Int [‘+ x ‘1]]

Kompozícia

V zápisu kompozície nie sú žiadne rozdiely medzi TIL a TIL-Scriptom. Na príklade môžete vidieť kompozíciu vyššie uvedeného príkladu uzáveru a trivializácie čísla 10:

[[\x:Int [‘+ x ‘1]] ‘10]

Exekúcia (prevedenie)

V TIL-Scriptu nie je špecifikovaná definícia Exekúcie (Dvojitej Exekúcie). Zápis konštrukcie exekúcie pomocou horného indexu („ ^{1}C “) bol v TIL-Scriptu nahradený zápisom („ $^{\wedge 1}$C “). Tak isto zápis konštrukcie dvojitej exekúcie je v TIL-Scriptu nahradený zápisom („ $^{\wedge 2}$C “).

Vytvorenie funkcií pomocou pomenovaných uzáverov

TIL-Script v sebe zahŕňa možnosť zostrojiť zadané funkcie pomocou pomenovaného uzáveru. K tomu sa používa kľúčové slovo *Def*. Napríklad pre funkciu, ktoré by bola pomenovaná ako *Succ* by sa v TIL-Scriptu použil zápis:

Def Succ := [\x:Int [‘ + x ‘1]]

Kvantifikátory

Pre zápis všeobecného kvantifikátoru (∀) je určený výraz *ForAll*, pre existenčný (∃) sa používa výraz *Exist*.

Logické spojky

Logické spojky sú v TIL-Scriptu reprezentované pomocou kľúčových slov: And, Or, Not, Implies, Equals

Zatiaľ čo v TIL môžeme vonkajšie zátvorky vynechať vždy, keď nemôže dôjsť k nedorozumeniu, v TIL-Scriptu ich nie je možné vynechať.

9.3 Zoznamy

Typ *zoznam* je z hľadiska programovacích jazykov veľmi dôležitý. Predstavuje sekvenciu objektov, ktorú je možno konštruovať zloženou TIL konštrukciou. Zoznamy boli zaradené do špecifikácie TIL-Scriptu z dôvodu častého využitia pri programovaní. Pre zoznam bolo vyhradené špeciálne slovo *list*. Jeho deklarácia vyzerá následovne:

list(typ1, typ2 ...)

Napríklad zoznam indivíduí sa vyjadruje týmto spôsobom:

list(Indiv) alebo *list(i)*.

Prehľadné zhrnutie všetkých rozdielov v syntaxi TIL a TIL-Scriptu je v nasledujúcej tabuľke.

Tabuľka č. 2: Porovnanie syntaxe TIL a TIL-Scriptu

TIL	TIL-Script	Popis
[,]	[,]	Zátvorky
^{0}P	*'P*	Trivializácia
/	/ (alebo : v uzáveru)	Definovanie typu
λ	\	Lambda
$P_{\omega\tau}$	P@wt	Použitie na svet a čas
^{2}P	$^{\wedge 2}$P	Dvojitá exekúcia
∀	ForAll	Všeobecný kvantifikátor
∃	Exist	Existenčný kvantifikátor
∧	And	Logická spojka AND
∨	Or	Logická spojka OR
¬	Not	Negácia
⊃	Implies	Implikácia
≡	Equals	Ekvivalencia

9.4 Príklady

Na nasledujúcich príkladoch môžete vidieť rozdiely v syntaxi medzi jazykmi TIL a TIL-Script.

Príklad č. 1:

Jozef je študent.

TIL:
λwλt [^{0}Študent$_{wt}$ ^{0}Jozef]

TIL-Script
[\w [\t [‘Študent@wt ‘Jozef]]]

Príklad č. 2:

Prezident Slovenskej republiky je ekonóm.

TIL:
λwλt [^{0}Ekonóm$_{wt}$ [^{0}Prezident$_{wt}$ ^{0}SR]]

TIL-Script
[\w [\t [‘Ekonóm@wt [‘Prezident@wt ‘SR]]]]

Príklad č. 3:

Karol Černý sa chce stať manželom Pavlíny Krátkej.

TIL:
λwλt [^{0}ChceSaStať$_{wt}$ ^{0}Karol_Černý λwλt [^{0}Manžel$_{wt}$ ^{0}Pavlíny_Krátkej]]

TIL-Script
[\w [\t [‘ChceSaStať@wt ‘Karol_Černý [\w[\t [‘Manžel@wt ‘Pavlíny_Krátkej]]]]]]

Príklad č. 4:

Všetky prvočísla sú nepárne.

TIL:
[$^0\forall$ λx [$^0\supset$ [^{0}Prvočíslo x] [^{0}Nepárne x]]]

TIL-Script
[‘ForAll [\x [‘Implies [‘Prvočíslo x] [‘Nepárne x]]]]

10. Popis riešenia

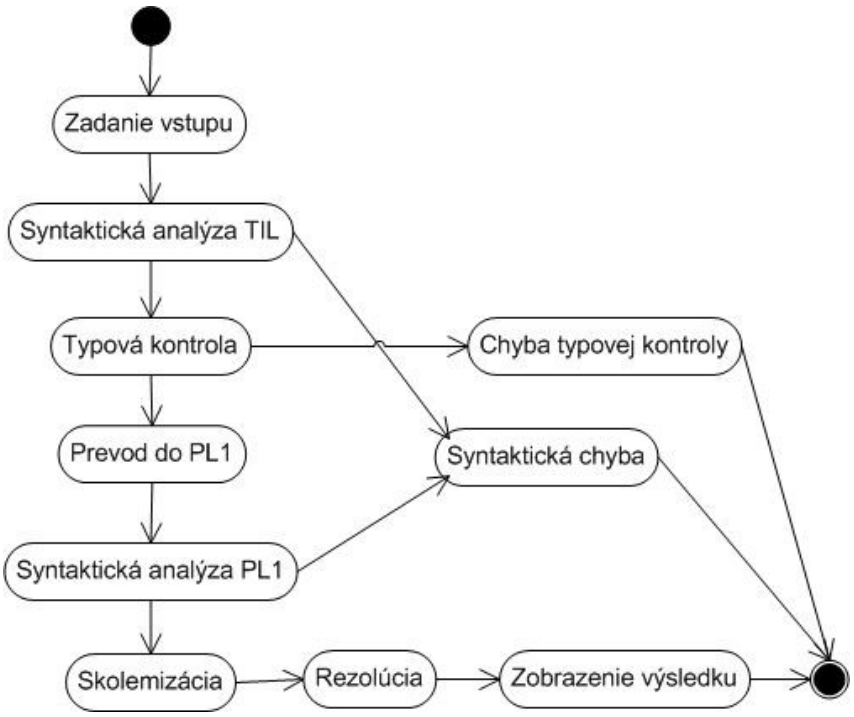
Po vymedzení podmnožiny tých vlastností TIL, ktoré sú preveditеľné do PL1 a priblížení si počítačovo spracovateľného jazyka TIL-Script sa možno pustiť do druhej časti tejto práce, ktorou je návrh a implementácia samotného inferenčného stroja. Program by mal fungovať tak, že užívateľ vloží vstupné konštrukcie (predpoklady a záver) zapísané pomocou TIL-Scriptu, počítač pomocou typovej kontroly overí formálnu správnosť zadaných konštrukcií. Vyhodnotí, či sú prevoditеľné do systému PL1 a v prípade úspechu ich prevedie a vráti výsledok (overí či záver vyplýva zo zadaných predpokladov, v prípade zadania iba záveru zistí, či sa jedná o tautológiu).

Najskôr bolo treba vyriešiť otázku vnútornej reprezentácie konštrukcií. Do programu vstupujú konštrukcie ako reťazce znakov, ktoré sú zapísané pomocou jazyka TIL-Script. Avšak pre ďalšiu prácu s konštrukciami to nie je ideálne riešenie, niektoré operácie by boli dosť zložité a algoritmicky náročné. Bolo preto potrebné nájsť inú, vhodnejšiu vnútornú reprezentáciu zadaných konštrukcií. A práve preto som sa priklonil k riešeniu pomocou stromovej reprezentácie konštrukcií, ktorá je síce náročnejšia na pamäťový priestor, ale podstatne uľahčí ďalšiu prácu.

Týmto sme vyriešili otázku vstupu a reprezentácie konštrukcií v programu. Ďalší krok pozostáva z overenia regulárnosti vstupného reťazca konštrukcie. Tu sa dostáva k slovu trieda *ParserFromTS*, ktorá obsahuje metódy pre overenie správneho zadania konštrukcií a typov, a metóda, ktorá vytvorí stromovú reprezentáciu vstupného reťazca. Následne prichádza na rad typová kontrola, ktorá overí, či jednotlivé podkonštrukcie, ktoré utvárajú výslednú konštrukciu formule odpovedajú typu celej konštrukcie.

Samotným prevodom konštrukcie do PL1 sa zaoberá trieda *PL1Convertor*. Výsledok prevodu sa predáva ako pole reťazcov triede *ConclusionProof*, ktorá predáva riadenie ďalším triedam (zaoberajúcim sa skolemizáciou a rezolúciou) a sprostredkováva výstupy.

Zjednodušený aktivitný diagram popisujúci algoritmus programu môžete vidieť na nasledujúcom obrázku.



Obrázok číslo 2.: Zjednodušený aktivitný diagram popisujúci program.

10.1 Overenie regulárnosti vstupných reťazcov

Aby bolo možné so vstupnými reťazcami (konštrukciami) pracovať, bolo potrebné overiť ich regulárnosť zápisu. Overenie regulárnosti vstupných reťazcov je prvým krokom, ktorý sa vykonáva pre každý vstup. Princíp spočíva v prevzatí pola vstupných reťazcov a následnom overení, či boli splnené vstupné podmienky (či bola zadaná aspoň jedna konštrukcia odpovedajúca záveru, a či všetky zadané konštrukcie sú formálne správne). Overenie regulárnosti vstupných reťazcov má na starosti trieda *ParserFromTS*, v ktorej sú implementované dve hlavné metódy: *check()* a *typovaKontrola()*. Metóda *check()* v sebe zahŕňa ďalšie metódy, ktoré plnia jednotlivé úlohy pri overení regulárnosti vstupných reťazcov.

Minimálnou požiadavkou na vstup je zadanie aspoň jednej konštrukcie (záveru) a definovanie typu tejto konštrukcie. Všeobecne, pre každú zadanú konštrukciu je potrebné definovať jej typ. Práve k tomu účelu bola implementovaná metóda *checkError()*, ktorá má na starosti overenie, či všetky zadané konštrukcie majú definovaný aj svoj typ.

Syntaktickú analýzu vstupných konštrukcií má na starosti metóda *typovaKontrolaError()*.
Vstupom sú tri reťazce – prvým je definovaná konštrukcia, druhým je definovaný typ tejto konštrukcie a tretím je reťazec slúžiaci k dodefinovaniu typov tých podkonštrukcií, ktoré neboli definované v samotnej konštrukcii. Definovanie typov ku všetkým podkonštrukciam je potrebné pre ďalšiu časť programu, ktorou je typová kontrola. Metóda *typovaKontrolaError()* sa odkazuje na ďalšie tri metódy: *analyzeTyp(),analyzeDefTyp(), analyzeFormule()*, ktoré majú na starosti analýzu týchto troch vstupných reťazcov.

Analýza typu formule

Overenie správnosti zadaného typu konštrukcie má na starosti metóda *analyzeTyp(),* ktorá načíta vstupný reťazec, a tento predá na spracovanie triede *Parser*. Tu sa celý vstupný reťazec rozparsuje pomocou oddeľovača, ktorým je čiarka, na samostatné typy a tieto sa predajú triede *Typ* na overenie, či je predaný typ definovaný v poli typov.

V poli typov sú definované tieto typy:

"Bool", "Indiv", "Time", "World", "Real", "Int", "String", "Any", "List", "Tuple", "*n"

V prípade, že predaný typ nebol v poli nájdený, vyvolá sa výnimka *AnalyzeException* a program vypíše chybové hlásenie s presním popisom, na ktorom riadku a na ktorej pozícii sa vyskytla chyba. V opačnom prípade program pokračuje analýzou definovaných typov.

Analýza definovaných typov

Program v sebe zahŕňa aj možnosť dodefinovania tých typov jednotlivých podkonštrukcií, ktoré neboli definované v samotnej konštrukcii. Overenie správneho zápisu týchto dodefinovaných podkonštrukcií a ich typov rieši metóda *analyzeDefTyp()*.
Správny zápis týchto dodatočne definovaných podkonštrukcií a ich typov je v tvare: p*odkonštrukcia:typPodkonštrukcie*. Typ sa od podkonštrukcie, pre ktorú je definovaný oddeľuje dvojbodkou. Vstupom môže byť viac podkonštrukcií, tieto od seba oddeľujeme prázdnou medzerou.

Vstupom je teda reťazec znakov (podkonštrukcií a ich typov). Program rekurzívne overuje správnosť zadaného reťazca, v prípade výskytu chyby vypíše chybové hlásenie s udaním pozície, na ktorej sa vyskytla chyba.

Analýza formule

Analýza konštrukcie je implementovaná metódou *analyzeFormule()*. Vstupom je konštrukcia reprezentovaná v podobe reťazca znakov. Metóda analyzuje vstupný reťazec, a na základe implementovaných pravidiel vytvára stromovú reprezentáciu tohto reťazca. Výstupom je teda binárny strom, ktorý odpovedá zadanej konštrukcii. Súčasťou metódy je aj kontrola správnosti zápisu konštrukcie. V prípade výskytu chyby program vypíše chybové hlásenie.

10.3 Stromová reprezentácia konštrukcie

Ako som už spomínal vyššie, práca s reťazcom znakov, reprezentujúcim zadanú konštrukciu (formulu) by bola dosť náročná. Bolo preto potrebné nájsť vhodnejšiu reprezentáciu zadaných konštrukcií, ktorá by uľahčila prácu. Práve z tohto dôvodu som sa priklonil k stromovej reprezentácii konštrukcie.

Stromová reprezentácia konštrukcie je založená na binárnom strome, ktorý je tvorený inštanciami triedy *TreeNode*. Každá inštancia tejto triedy v podstate predstavuje jeden vrchol stromu. Jednotlivé uzly stromu sa rozdeľujú na binárne a unárne.

Unárne uzly majú vždy iba jeden ukazateľ na ďalší podstrom. Tieto unárne uzly sú implementované tak, že využívajú len pravý ukazateľ na následníka.

Binárne uzly sú definované ľavým a pravým listom, majú teda dva ukazatele – a to na ľavý a na pravý podstrom.

Všeobecne môžeme povedať, že binárne uzly sú zastúpením operátorov: konjunkcia, disjunkcia, implikácia, ekvivalencia. Všetky ostatné znaky a podkonštrukcie (ako sú hranaté zátvorky, kvantifikátory, trivializácie, premenné, lambda abstrakcie, atď.) tvoria uzly unárne.

Jednotlivé listy potom reprezentujú textové reťazce. (jednotlivé podkonštrukcie, zátvorky a pod.)

Samotná tvorba týchto stromov sa vykonáva v metóde *analyzeFormule()*.

Na nasledujúcich príkladoch názorne ukážem stromovú reprezentáciu konštrukcií. Uvediem dva príklady, v prvom bude strom tvorený iba unárnymi uzlami, v druhom využijem aj uzly binárne.

Príklad 1:

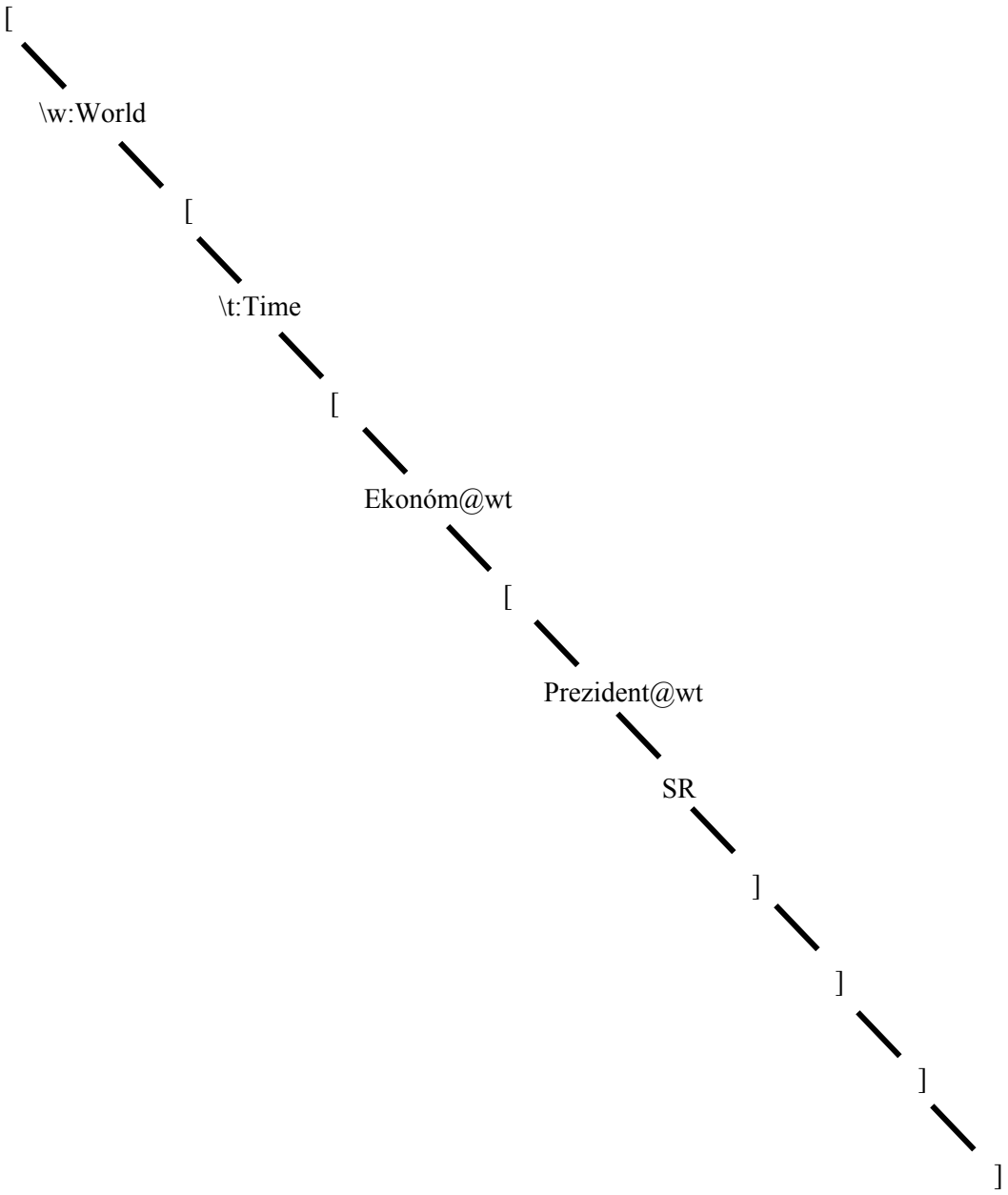
Prirodzený jazyk:

Prezident Slovenskej republiky je ekonóm.

Zápis formule v TIL-Scriptu:

[\w:World [\t:Time [‘Ekonóm@wt [‘Prezident@wt ‘SR]]]]

Stromová reprezentácia formule:



Príklad 2:

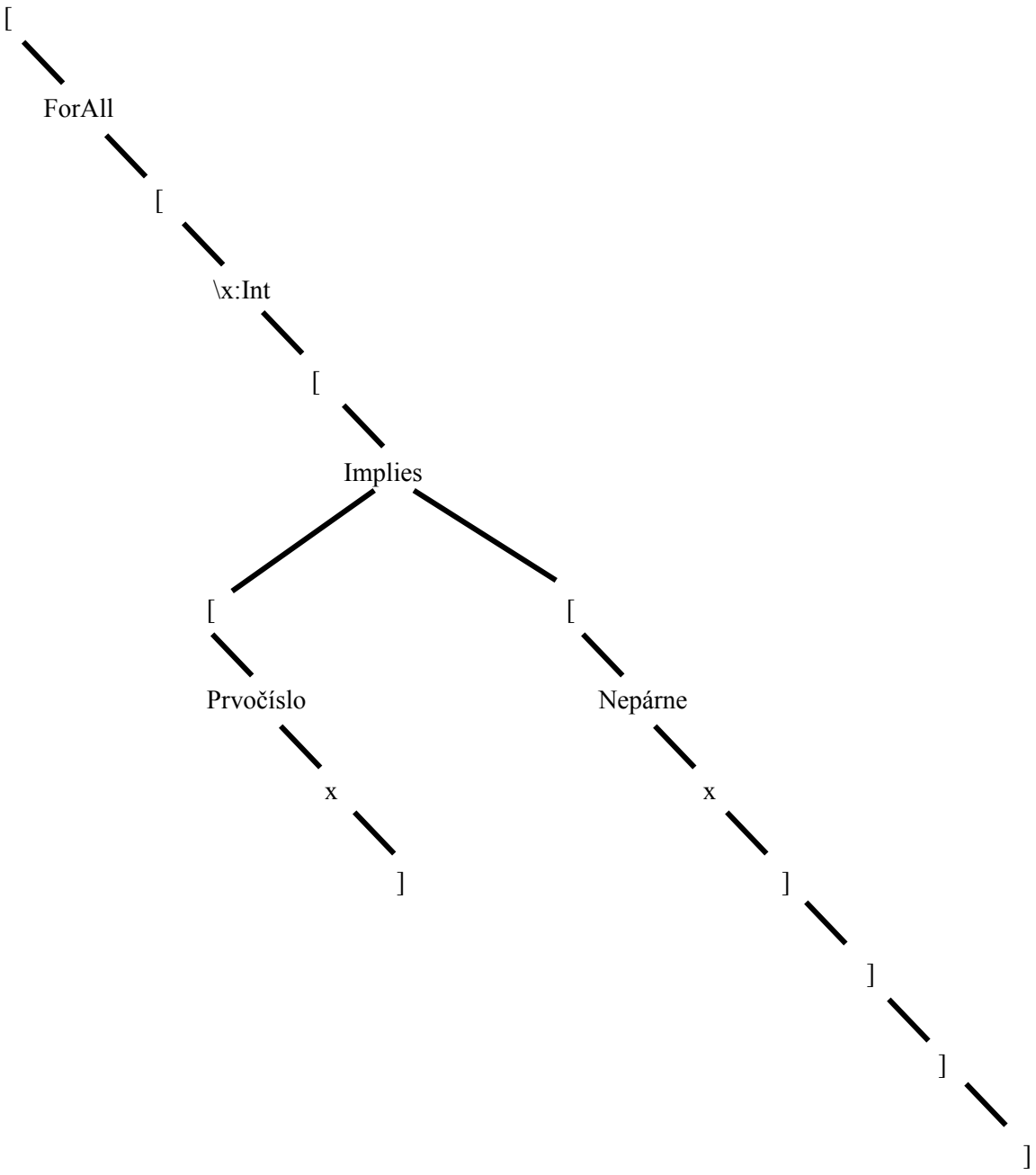
Prirodzený jazyk:

Všetky prvočísla sú nepárne

Zápis formule v TIL-Scriptu:

[‘ForAll [\x:Int [‘Implies [‘Prvočíslo x] [‘Nepárne x]]]]

Stromová reprezentácia formule:



10.2 Reprezentácia symbolov TIL a PL1

Pri spracovávaní konštrukcií a ich následnom prevádzaní na symboly predikátovej logiky prvého radu narážame na symboly, ako napríklad: negácia, kvantifikátory, symboly logických spojok, ktoré nie sú prirodzenou súčasťou štandardných fontov. Bolo preto potrebné nájsť vhodnú reprezentáciu týchto symbolov v programe. Keďže vstupom do programu sú konštrukcie TIL a výstupom reťazce znakov zapísané pomocou PL1, rozdelíme si túto reprezentáciu na dve časti: vonkajšiu – ktorá bude reprezentovať vstupné konštrukcie TIL a vnútornú, ktorá bude reprezentovať symboly PL1. Reprezentáciu symbolov používaných v programe môžete vidieť v nasledujúcej tabuľke:

Tabuľka č. 3: Reprezentácia symbolov v programu

Vonkajšia reprezentácia konštrukcií TIL	Vnútorná reprezentácia symbolov v PL1	Popis
[,]	(,)	Zátvorky
FORALL	!	Všeobecný kvantifikátor: ∀
EXIST	1	Existenčný kvantifikátor: ∃
AND	*	Logická spojka AND: ∧
OR	+	Logická spojka OR: ∨
NOT	-	Negácia ¬
IMPL	>	Implikácia ⊃
EQL	<>	Ekvivalencia ≡
‘	nedefinovaná pre PL1	Trivializácia
\	nedefinovaná pre PL1	Lambda
P@wt	nedefinované pre PL1	Použitie na svet a čas

10.4 Typová kontrola

Typová kontrola slúži na overenie, či zadaná vstupná konštrukcia odpovedá typu, ktorý bol definovaný k tejto konštrukcii. Vykonáva sa rekurzívne pre každú konštrukciu, ktorá bola zadaná na vstupu. Typová kontrola je implementovaná triedou *TypeCheck*. Pracuje sa už s vytvorenou stromovou reprezentáciou vstupnej konštrukcie. Princíp spočíva v rekurzívnom prechádzaní stromu od najnižšieho listu až po koreňový uzol, pričom sa kontrolujú typy uložené v jednotlivých uzloch. Pre konštrukcie, ktoré nemajú v uzlu uložený svoj typ, sa tieto získavajú z Mapy Stringov, do ktorej boli uložené pomocou metódy *analyzeDefTyp()*. V každom uzlu sa overuje možnosť odčítania typov od svojho nadradeného uzlu.

Výsledný typ stromovej reprezentácie konštrukcie sa nakoniec porovnáva v triede *ParserFromTS*, v metóde *typovaKontrola()* s typom, ktorý bol pred danú konštrukciu definovaný na vstupu. V prípade nezhody program vypíše chybové hlásenie.

10.5 Prevod do PL1

Po úspešnom vykonaní typovej kontroly sa dostáva k slovu samotný prevod konštrukcie do syntakticky regulárneho zápisu predikátovej logiky prvého radu. Celý algoritmus prevodu je implementovaný v triede *PL1Convertor*.

Hlavnú úlohu pri prevode zohráva metóda *parse()*, ktorá rekurzívne prechádza strom vytvorený z konštrukcie zadanej na vstupe a podľa implementovaných podmienok prevádza jednotlivé výrazy (symboly) uložené v týchto uzloch.

Prevod jednoduchých symbolov, ako sú zátvorky, alebo konštrukcií reprezentujúcich logické spojky a kvantifikátory nebol zložitý. Stačilo tieto konštrukcie (symboly) nahradiť vhodným znakom reprezentujúcim túto konštrukciu (symbol) v PL1.
Problém však nastal pri prevode konštrukcií na predikátové alebo funkčné symboly. Bolo potrebné pamätať si jednotlivé konštrukcie uložené na uzloch, aby bolo možné, v prípade, že by sa takáto konštrukcia vyskytla znovu, previesť ju na rovnaký predikátový alebo funkčný symbol. K tomuto účelu boli vytvorené štyri mapy(hešovacie tabuľky), jedna pre predikátové symboly, druhá pre premenné, tretia pre funkcie a štvrtá pre konštanty.
Algoritmus prevodu funguje tak, že program načíta konštrukciu (symbol) uloženú v uzle, overí či sa jedná o zátvorku, logickú spojku alebo kvantifikátor, a ak ani jedna z podmienok nevyhovie, algoritmus pokračuje rozhodovaním, aký predikátový alebo funkčný symbol danej konštrukcii, uloženej v uzlu, priradiť. Toto rozhodovanie sa vykonáva na základe podmienok, ktoré zohľadňujú typ, ktorým konštrukcia uložená v uzle (s ktorým sa aktuálne pracuje) disponuje. Po vykonaní podmienky rozhodne, do ktorej mapy (hešovacej tabuľky) túto konštrukciu (nachádzajúcu sa v aktuálnom uzle) uloží a do výstupného reťazca zapíše odpovedajúci predikátový alebo funkčný symbol. V prípade, že mapa už túto konštrukciu obsahuje, program ju tam duplicitne nezapíše, ale len zistí na ktorej pozícii sa nachádza a do výstupného reťazca zapíše odpovedajúci symbol (predikátový alebo funkčný).
Pre jednoduché pochopenie ešte vysvetlím princíp máp. Do máp sa na jednotlivé pozície ukladajú konštrukcie z uzlov stromu. Každá pozícia v mape je reprezentovaná symbolom, ktorý sa predáva do výstupného reťazca. Symboly sú definované pomocou písmen, v rozmedzí *a – z*. V prípade máp reprezentujúcich predikátové symboly sú to písmena A – Z. U máp reprezentujúcich premenné sú to písmená a – z. U máp reprezentujúcich funkcie sú to symboly a() – z().

10.6 Rezolúcia

Súčasťou tejto diplomovej práce je aj návrh a implementácia samotnej rezolučnej metódy, pre dokazovanie platnosti úsudkov v predikátovej logike prvého radu s využitím TIL ako špecifikačný jazyk. V programe využívam algoritmus rezolučnej metódy vytvorený Petrom Vyletělkom, ako súčasť jeho bakalárskej práce. Tento je implementovaný v programovacom jazyku Java a pozostáva z dvoch hlavných častí – z prevodu formule do klauzulárnej formy a z algoritmu obecnej rezolučnej metódy.

Výsledný reťazec prevodu konštrukcie TIL z triedy *PL1Convertor* sa ukladá do pola reťazcov, ktoré je následne predané na ďalšie spracovávanie triede *ConclusionProof*, ktorá zaisťuje predávanie riadenia ďalším triedam (zaoberajúcich sa rezolúciou a skolemizáciou) a sprostredkováva výstupy. V ďalšom kroku sa tieto reťazce z pola doplnia o zátvorky, ktoré uzavrú celý reťazec, v prípade záveru sa pred uzátvorkovanie pridá negácia. Takto uzátvorkované reťazce vstupujú do syntaktická analýzy PL1. V prípade že reťazec, ktorý je výsledkom prevodu konštrukcie TIL do formule PL1 nezodpovedá definovanej gramatike, program vyhodí chybové hlásenie. Ak syntaktická analýza prebehne bez chyby, tieto upravené reťazce sa následne preložia konjunkciami a výsledok sa uloží ako jeden vstupný reťazec k ďalšiemu spracovávaniu – nasleduje skolemizácia.

Skolemizácia

Táto časť algoritmu rieši prevod už preloženého reťazca do klauzulárnej formy, na ktorú budeme potom aplikovať rezolučnú metódu. Skladá sa z deviatich krokov a je implementovaná v triede *Skolemizator*.

Prvým krokom skolemizácie je utvorenie existenčného uzáveru. Cieľom tohto kroku je dosiahnuť toho, aby vo výslednej formuli nebola žiadna voľná premenná. Pre tento krok bola vytvorená vlastná trieda *ExistenceEnclosure*. Princíp spočíva v rekurzívnom prechádzaní stromu, kde v v jednotlivých listoch stromu dochádza k porovnávaniu premenných na existenciu definovaného kvantifikátoru k tejto premennej. Ak program narazí na premennú, ktorá nemá definovaný kvantifikátor uloží si ju do zoznamu a po prejdení celého stromu pridá na začiatok formule existenčné kvantifikátory k týmto premenným.

Ďalšie dva kroky skolemizácie – eliminácia nadbytočných kvantifikátorov a premenovanie premenných, sa vykonávajú spoločne počas jedného prechodu stromom.

Štvrtým krokom skolemizácie je eliminácia spojok implikácie a ekvivalencie. K tomuto bola vytvorená metóda *connectivesElimination(Node root)*.

Piaty krok skolemizácie pozostáva z presunu negácie doprava. K tomuto slúži metóda *negationToTheRight(Node root)*.

Šiestym krokom skolemizácie je presun kvantifikátorov doprava. K tomu bola implementovaná metóda *quantifiersToTheRight(Node root)*.

Siedmy krok skolemizácie spočíva v eliminácii existenčných kvantifikátorov. K tomuto bola implementovaná trieda *Skolemization*.

Krok číslo osem má na starosti presun všeobecných kvantifikátorov doľava a je implementovaný pomocou metódy *generalQuantifiersToTheLeft(Node root)*.

Záverečný deviaty krok skolemizácie má na starosti upravenie formule na konjunktívny normálny tvar použitím distributívnych zákonov. Je implementovaný v metóde *useDistributionLaws(root)*.

Vykonaním týchto deviatich krokov skolemizácie dostávame akceptovateľný tvar, ktorý vstupuje do samotnej rezolúcie. Celý algoritmus rezolúcie je implementovaný v triede *Resolution*. Algoritmus je naprogramovaný tak, že postupne porovnáva všetky klauzule. Vždy zoberie jednu klauzulu a porovná ju so všetkými ostatnými. Takto postupuje až kým aplikovaním rezolučného pravidla nedostane prázdnu klauzulu, alebo kým neporovná posledné dve klauzule (aj keď nedostane prázdnu klauzulu).

V prípade že získame prázdnu klauzulu, celý algoritmus ukončíme s tým, že získaná klauzule je nesplniteľná, čiže jej negácia je vždy pravdivá.

V opačnom prípade (keď porovnáme všetky klauzule a nezískame prázdnu klauzulu) môžeme o prijatej formuli povedať že je splniteľná a negácia tejto formule nie je tautológia.

10.7 Implementácia

Celý program je implementovaný ako Java Applet v programovacom jazyku Java, vo verzii 1.6. Aplikácia je určená pre platformu Microsoft Windows (kompatibilná s verziami Windows XP a vyššie).

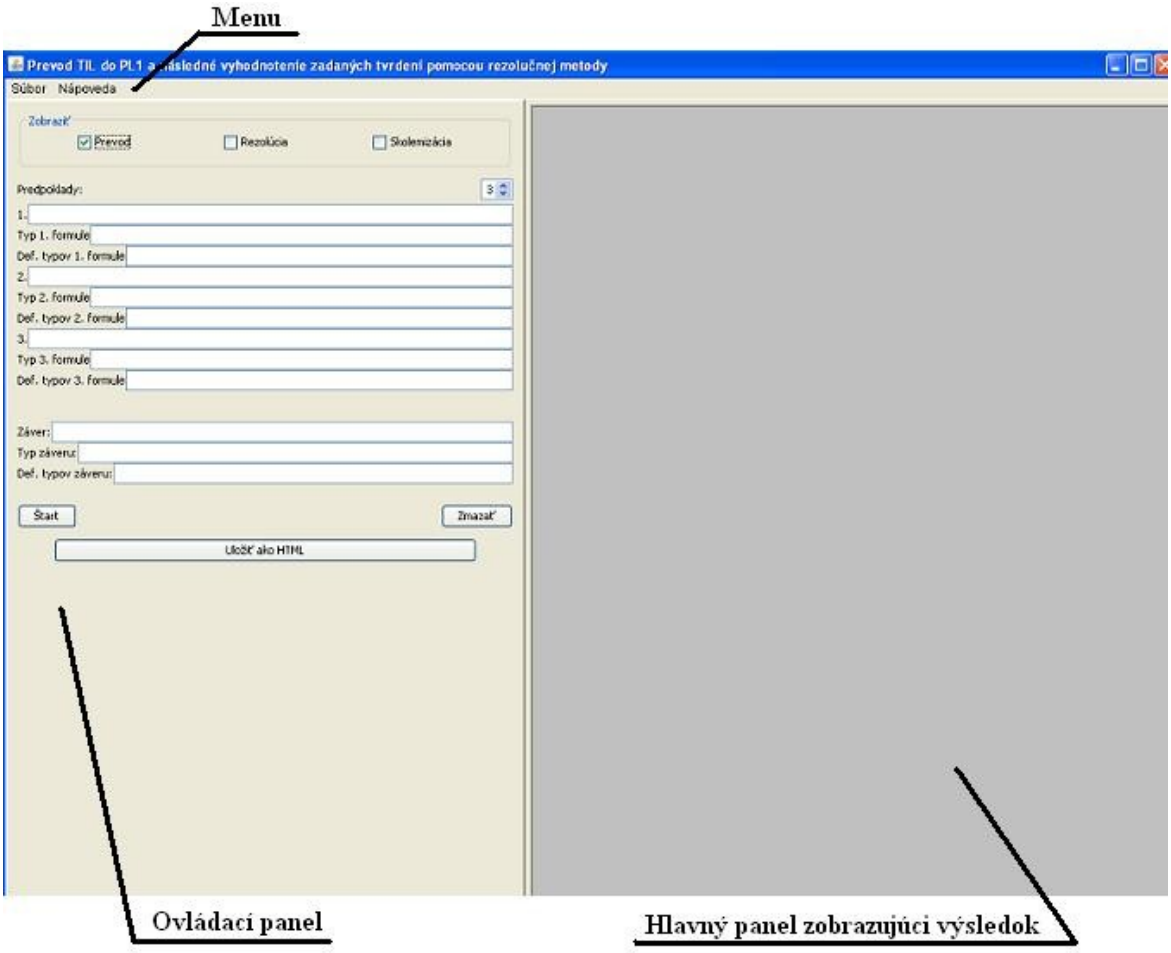
11. Užívateľská dokumentácia

11.1 Inštalácia a spustenie

Tato aplikácia bola vytvorená v programovacom jazyku Java vo verzii 1.6. Pre jej spustenie je potrebné mať v počítači nainštalované prostredie, umožňujúce spúšťanie Javovských aplikácií. Takéto prostredie je voľné prístupné na stránkach:
http://www.oracle.com/technetwork/java/index.html.
Samotná aplikácia sa potom spúšťa pomocou súboru *run.bat*.

11.2 Popis

Po úspešnom spustení programu sa vám objaví hlavné okno celej aplikácie. Skladá sa z troch častí: *Menu, Ovládací panel, Hlavný panel zobrazujúci výsledok*.



Obrázok číslo 3.: Užívateľské okno aplikácie

Menu

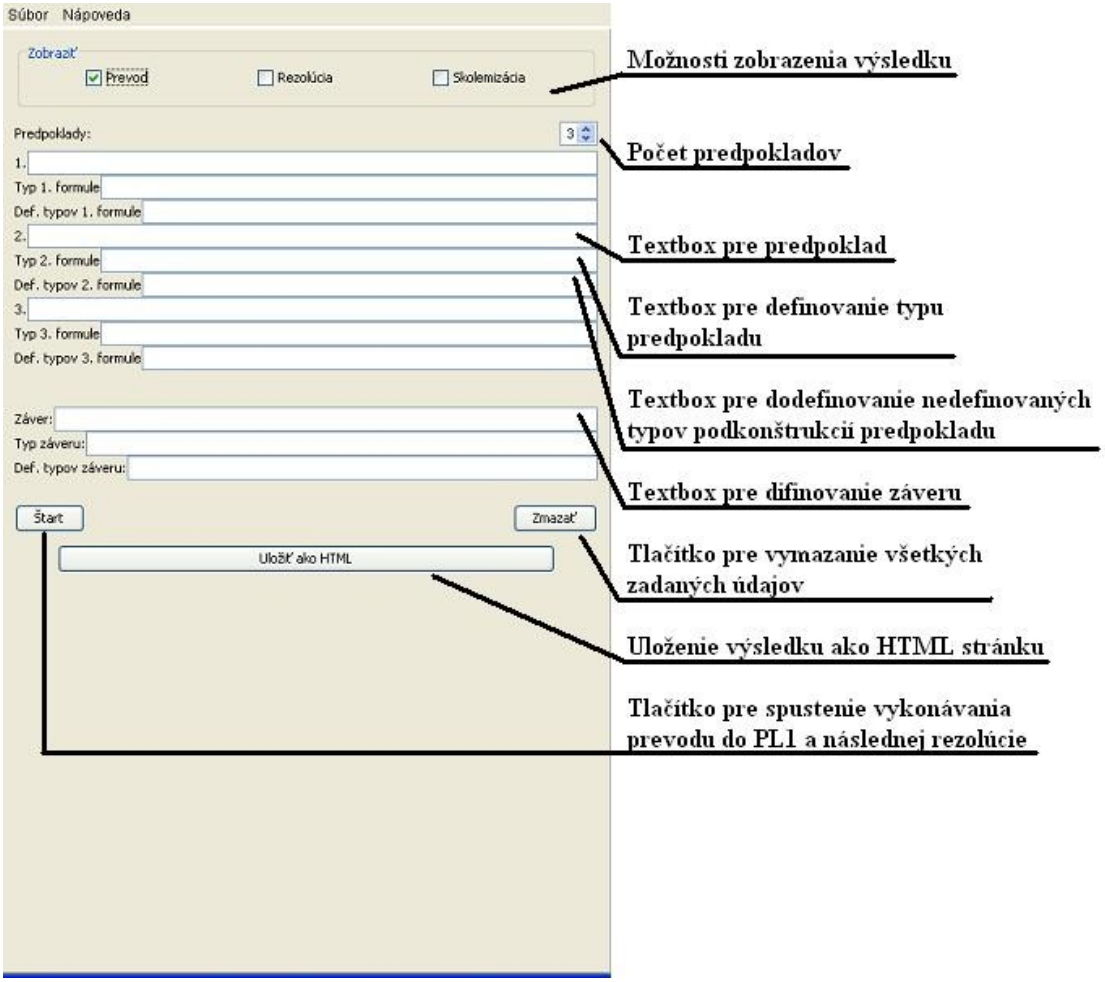
Menu obsahuje dve položky, a to: *Súbor* a *Nápoveda.*
V menu *Súbor* je položka *Uložiť výsledok*, ktorá slúži na uloženie výsledku, zobrazeného v hlavnom panely, do súboru. Výsledok sa ukladá do textového súboru.

Položka *Koniec* slúži na ukončenie celej aplikácie.

V menu Nápoveda sú dve položky, a to: *Použitie programu, O programu*. Položka *O programu* v sebe skrýva základné informácie o tomto programu, autora, dátum vzniku a pod. Položka *Použitie programu* v sebe zahŕňa základné pravidlá a informácie, ako používať tento program, akým spôsobom zapisovať vstupné formule a pod.

Ovládací Panel

Hlavným ovládacím prvkom celej aplikácie je tento ovládací panel. Slúži hlavne na zadávanie jednotlivých konštrukcií (predpokladov a záveru). K tomu slúžia textové pola pre predpoklady (sú očíslované číslicami). Ku každému predpokladu musí byť definovaný aj typ tohto predpokladu. K tomu slúži textové pole s názvom: *Typ 1. formule*, kde toto číslo udáva číslo predpokladu. Pre dodefinovanie typov podkonštrukcií formulí, ktoré nie sú definované v samotnej konštrukcii (formuli) slúži textové pole s názvom: *Def. typov*.
Pod všetkými predpokladmi sa nachádza textové pole pre definovanie záveru, jeho typu a dodefinovaniu nedefinovaných podkonštrukcií záveru. Definovanie záveru je povinné!



Obrázok číslo 4.: Ovládací panel aplikácie

Napravo od nápisu *Predpoklady:* sa nachádza textové pole so šípkami a číslom, určujúcim aktuálny počet možných predpokladov. Implicitne sú nastavené štyri predpoklady. Toto číslo je možné meniť v rozmedzí 3 – 9 predpokladov.

V hornej časti ovládacieho panelu sa nachádzajú možnosti pre nastavenie zobrazenia výsledku. Zaškrtnutím určujeme, ktoré všetky časti priebehu riešenia sa majú zobraziť v hlavnom panely.

V spodnej časti ovládacieho panelu sa nachádzajú tri tlačidla. Tlačidlo *Štart* slúži na spustenie algoritmu prevodu do PL1 a následne dokazovanie platnosti úsudkov. Druhé tlačidlo *Zmazať* slúži na zmazanie všetkých textových polí (predpokladov, typov a dodefinovaných typov). Posledné tlačidlo: *Uložiť ako HTML* slúži na uloženie zadania a výsledku do formátu HTML stránky.

11.3 Použitie programu

Tento program slúži na prevádzanie konštrukcií (formulí) zadaných pomocou TIL-Scriptu do predikátovej logiky prvého radu a následne dokazovanie platnosti úsudkov a tautológií pomocou obecnej rezolučnej metódy. Minimálnou požiadavkou na spustenie programu je zadanie záveru a jeho typu. V prípade, že nie sú vyplnené žiadne predpoklady, iba záver, program prevedie vyplnený záver do PL1 a overuje, či sa jedná o tautológiu.

Použitie programu je veľmi jednoduché. Do textových polí pre predpoklady vypíšte jednotlivé predpoklady. Každý zadaný predpoklad musí mať definovaný svoj typ, ktorý zadajte do textového pola *Typ formule*. Po vyplnení všetkých predpokladov je potrebné doplniť záver do textového pola *Záver* a jeho typ do textového pola *Typ záveru*.

Teraz ešte zaškrtnite, čo všetko chcete vo výsledku zobraziť. K tomu slúžia zaškrtávacie políčka v hornej časti ovládacieho panelu – *Prevod*, *Rezolúcia* a *Skolemizácia*. Po zaškrtnutí *Rezolúcie* sa vo výsledku zobrazí celý priebeh aplikácie rezolučného pravidla. Po zaškrtnutí *Skolemizácie* sa vo výsledku zobrazí i postupný prevod formule na klausulárny tvar.

Po splnení všetkých predchádzajúcich krokov stačí kliknúť na tlačidlo *Štart*. V prípade že sme všetky políčka vyplnili správne, zobrazí sa vám v hlavnom paneli výsledok vášho zadania. Ak pri zadávaní zabudneme vyplniť niektoré potrebné políčko, program na to upozorní informačnou správou. V prípade zle definovanej konštrukcie (formule), typov alebo dodefinovaných typov, program vypíše chybové hlásenie.

V prípade nutnosti vymazania všetkých zadaných polí stačí kliknúť na tlačidlo *Zmazať*.

Pravidla pre zápis formulí

Každá formula musí začínať a končiť hranatou zátvorkou. Pri zápise formulí je možné používať iba hranaté zátvorky. Výraz Lambda je nahradený spätným lomítkom: „\“. Trivializácia sa znázorňuje pomocou apostrofu „ ' “. Jednotlivé podkonštrukcie v zátvorkách (kompozícií) od seba oddeľujte prázdnou medzerou

Definovanie typov formulí:

Jednotlivé typy od seba oddeľuje čiarkou. Je možné definovať iba tieto typy: Bool, Indiv, Time, World, Real, Int, String, Any, List, Tuple, *n. Pre aplikáciu na možný svet a čas uzatvorte typy guľatými zátvorkami, nasledovanými výrazom @wt.

Príklady definovania typov: Bool,Time,World
(Bool,Indiv)@wt

Definovanie nedefinovaných typov podkonštrukcií:

Typy podkonštrukcií definujte tak, že zadáte výraz (podkonštrukciu) nasledovaný dvojbodkou a za dvojbodkou zadajte typ výrazu. Jednotlivé výrazy oddeľujte prázdnou medzerou.

Príklad definovania nedefinovaných typov podkonštrukcií:

Peter:Indiv Ján:Indiv Študent:(Bool,Indiv)@wt

Logické spojky:

AND - konjunkcia
OR - disjunkcia
IMPL - implikácia
EQL - ekvivalencia
NOT - negácia

Kvantifikátory:

FORALL - všeobecný
EXIST - existenčný

Príklad zápisu formule:

['FORALL[\x:Indiv['AND['Student@wt x]['Usilovny@wt x]]]]

12. Záver

Transparentná intenzionálna logika patrí v dnešnej dobe medzi najsilnejšie logiky. Svoje uplatnenie nachádza predovšetkým v tých miestach, kde je analýza tvrdení pomocou klasických systémov príliš slabá. Sú to práve také tvrdenia prirodzeného jazyka, ktoré sú závislé na možných svetoch.

Cieľom tejto práce bolo vymedziť takú podmnožinu TIL, ktorú je možné previesť do predikátovej logiky prvého radu a následne navrhnúť a naimplementovať inferenčný stroj, ktorý bude schopný prevádzať konštrukcie z TIL do PL1, a overovať pomocou rezolučnej metódy platnosť úsudkov.

Úspešne bola vymedzená podmnožina TIL, ktorá je previediteľná do PL1 a následne naimplementovaný program prevodu a algoritmus overovania platnosti úsudkov pomocou obecnej rezolučnej metódy. Napriek tomu, že neboli implementované všetky vlastnosti TIL (ako napríklad parcialita), riešenie dotazov na úrovni logiky prvého radu bolo úspešné.

Táto práca bola pre mňa jednoznačne prínosná. Využil som svoje doteraz nadobudnuté vedomosti hlavne z oblasti matematickej logiky, no i napriek tomu som bol donútený si mnoho vecí naštudovať samostatne. Rozšíril som svoje vedomosti o nový logický systém TIL. Skombinovaním nadobudnutých vedomostí som bol schopný vytvoriť inferenčný stroj, ktorý je výsledkom tejto práce. Verím že bude pre spoločnosť prínosom.

Vytvorený inferenčný stroj ponúka základné funkcie pre jeho využitie. Do budúcna je možné ho rozšíriť o ďalšie zaujímavé funkcie. Nebolo by na škodu zaoberať sa otázkou výkonnosti, ako napríklad rýchlosťou spracovávania vstupných dát.

Literatúra

[1] Duží, Marie. *Matematická logika*. VŠB-TU Ostrava, 2003.

[2] Duží, Marie. *Principy logické analýzy jazyka*. VŠB-TU Ostrava, 2003.

[3] DUŽÍ, Marie – JESPERSEN, Bjorn – MATERNA, Pavel. *Procedural Semantics for Hyperintensional Logic*: foundations and applications of transparent intensional logic. 1. vyd. Dordrecht:Springer, 2010. 552 s. ISBN 978-90-481-8811-6.

[4] Duží, M., Číhalová, M., Ciprich, N., Menšík, M.: Agents' reasoning using TIL-Script and Prolog.. In 19th European-Japanese Conference on Information Modelling and Knowledge Bases. Ed. T. Tokuda, Y. Kiyoki, H. Jaakkola, T. Welzer Družovec, Slovenia:University of Maribor, 2009, 137-156, ISBN 978-961-248-162-9